ISA Wars: Understanding the Relevance of ISA being RISC or CISC to Performance, Power, and Energy on Modern Architectures

EMILY BLEM, JAIKRISHNAN MENON, THIRUVENGADAM VIJAYARAGHAVAN, and KARTHIKEYAN SANKARALINGAM, University of Wisconsin - Madison

RISC versus CISC wars raged in the 1980s when chip area and processor design complexity were the primary constraints and desktops and servers exclusively dominated the computing landscape. Today, energy and power are the primary design constraints and the computing landscape is significantly different: Growth in tablets and smartphones running ARM (a RISC ISA) is surpassing that of desktops and laptops running x86 (a CISC ISA). Furthermore, the traditionally low-power ARM ISA is entering the high-performance server market, while the traditionally high-performance x86 ISA is entering the mobile low-power device market. Thus, the question of whether ISA plays an intrinsic role in performance or energy efficiency is becoming important again, and we seek to answer this question through a detailed measurement-based study on real hardware running real applications. We analyze measurements on seven platforms spanning three ISAs (MIPS, ARM, and x86) over workloads spanning mobile, desktop, and server computing. Our methodical investigation demonstrates the role of ISA in modern microprocessors' performance and energy efficiency. We find that ARM, MIPS, and x86 processors are simply engineering design points optimized for different levels of performance, and there is nothing fundamentally more energy efficient in one ISA class or the other. The ISA being RISC or CISC seems irrelevant.

Categories and Subject Descriptors: C.0 [**General**]: Hardware/Software Interfaces, Instruction Set Design, System Architectures

General Terms: Design, Measurement, Performance

Additional Key Words and Phrases: Power, energy efficiency, technology scaling

ACM Reference Format:
Emily Blem, Jaikrishnan Menon, Thiruvengadam Vijayaraghavan, and Karthikeyan Sankaralingam. 2015. ISA wars: Understanding the relevance of ISA being RISC or CISC to performance, power, and energy on modern architectures. ACM Trans. Comput. Syst. 33, 1, Article 3 (March 2015), 34 pages.
DOI: http://dx.doi.org/10.1145/2699682

1. INTRODUCTION

The question of ISA design, and specifically RISC versus CISC ISA, was an important concern in the 1980s and 1990s when chip area and processor design complexity were the primary constraints [Patterson and Ditzel 1980; Colwell et al. 1985; Flynn et al. 1987; Bhandarkar and Clark 1991]. It is questionable if the debate was settled in terms of technical issues. Regardless, both flourished commercially throughout the 1980s and 1990s. In the past decade, the ARM and MIPS ISAs (RISC ISAs) have

This work is supported by NSF grants CCF-0845751, CCF-0917238, and CNS-0917213, and the Cisco Systems Distinguished Graduate Fellowship.

Authors' addresses: E. Blem, Google Inc., 1600 Amphitheatre Parkway, Mountain View, CA 94043; email: emilyblem@gmail.com; J. Menon, T. Vijayaraghavan, and K. Sankaralingam, Department of Computer Sciences, University of Wisconsin-Madison, 1210 West Dayton Street, Madison, WI 53706; emails: jmenon86@gmail.com, {thiruven, karu}@cs.wisc.edu.

DOI: http://dx.doi.org/10.1145/2699682

dominated mobile and low-power embedded computing domains and the x86 ISA (a CISC ISA) has dominated desktops and servers.

Recent trends raise the question of the role of ISA and make a case for revisiting the RISC versus CISC question. First, the computing landscape has quite radically changed from when the previous studies were done. Rather than being exclusively desktops and servers, today's computing landscape is significantly shaped by smartphones and tablets. Second, whereas area and chip design complexity were previously the primary constraints, energy and power constraints now dominate. Third, from a commercial standpoint, both ISAs are appearing in new markets: ARM-based servers for energy efficiency and x86-based mobile and low-power devices for high performance. As a recent example, the Quark line of x86-based designs are entering the traditionally RISC microcontroller regime. Thus, the question of whether ISA plays a role in performance, power, or energy efficiency is once again important.

Related Work. Early ISA studies are instructive but miss key changes in today's microprocessors and design constraints that have shifted the ISA's effect. We review previous comparisons in chronological order and observe that all prior comprehensive ISA studies considering commercially implemented processors focused *exclusively on performance.*

Bhandarkar and Clark compared the MIPS and VAX ISA by comparing the M/2000 to the Digital VAX 8700 implementations [Bhandarkar and Clark 1991] and concluded: "RISC as exemplified by MIPS provides a significant processor performance advantage." In another study in 1995, Bhandarkar compared the Pentium-Pro to the Alpha 21164 [Bhandarkar 1997], again focused exclusively on performance and concluded: "the Pentium Pro processor achieves 80% to 90% of the performance of the Alpha 21164... It uses an aggressive out-of-order design to overcome the instruction set level limitations of a CISC architecture. On floating-point intensive benchmarks, the Alpha 21164 does achieve over twice the performance of the Pentium Pro processor." Consensus had grown that RISC and CISC ISAs had fundamental differences that led to performance gaps that required aggressive microarchitecture optimization for CISC that only partially bridged the gap.

Isen et al. [2009] compared the performance of Power5+ to Intel Woodcrest considering SPEC benchmarks and concluded that x86 matches the POWER ISA. The consensus was that "with aggressive microarchitectural techniques for ILP, CISC and RISC ISAs can be implemented to yield very similar performance."

Many informal studies in recent years claim the x86's "crufty" CISC ISA incurs many power overheads and attribute the ARM processor's power efficiency to the ISA.[1] These studies suggest that the microarchitecture optimizations from the past decades have led to RISC and CISC cores with similar performance but that the power overheads of CISC are intractable.

In light of the ISA studies from decades past, the significantly modified computing landscape, and the seemingly vastly different power consumption of RISC implementations (ARM: 1–2W, MIPS: 1–4W) to CISC implementations (x86: 5–36W), we feel there is need to revisit this debate with a rigorous methodology. Specifically, considering the multipronged importance of the metrics of power, energy, and performance, we need to compare RISC to CISC on those three metrics. Macro-op cracking and decades of research in high-performance microarchitecture techniques and compiler optimizations seemingly help overcome x86's performance and code-effectiveness bottlenecks, but

[1] ARM On Ubuntu 12.04 LTS Battling Intel x86 (http://www.phoronix.com/scan.php?page=article&item=ubuntu_1204_armfeb&num=1). The ARM vs x86 Wars Have Begun: In-Depth Power Analysis of Atom, Krait & Cortex A15 (http://www.anandtech.com/show/6536/arm-vs-x86-the-real-showdown/).

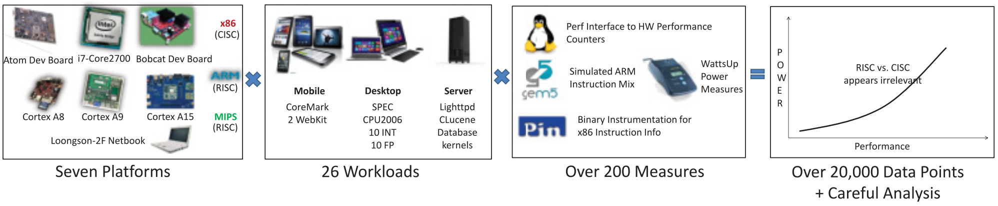


Fig. 1. Summary of approach.

these approaches are not free. The crux of our analysis is the following: *After decades of research to mitigate CISC performance overheads, do the new approaches introduce fundamental energy inefficiencies?*

Challenges. Any ISA study faces challenges in separating out the multiple implementation factors that are orthogonal to the ISA from the factors that are influenced or driven by the ISA. ISA-independent factors include chip process technology node, device optimization (high-performance, low-power, or low-standby power transistors), memory bandwidth, I/O device effects, operating system, compiler, and workloads executed. These issues are exacerbated when considering energy measurements/analysis, since chips implementing an ISA sit on boards and separating out chip energy from board energy presents additional challenges. Furthermore, some microarchitecture features may be required by the ISA, whereas others may be dictated by performance and application domain targets that are ISA-independent.

To separate out the implementation and ISA effects, we consider multiple chips for each ISA with similar microarchitectures, use established technology models to separate out the technology impact, use the same operating system and compiler front-end on all chips, and construct workloads that do not rely significantly on the operating system. Figure 1 presents an overview of our approach: the seven platforms, 26 workloads, and set of measures collected for each workload on each platform. We analyzed one MIPS implementation (Loongson), three ARM implementations (Cortex-A8, Cortex-A9, and Cortex-A15), and three x86 implementations (Atom, Bobcat, and Sandybridge i7). These implementations span diverse ISAs and, within each ISA, span diverse microarchitectures.

We present an exhaustive and rigorous analysis using workloads that span smartphone, desktop, and server applications. In our study, we are primarily interested in *whether and, if so, how the ISA being RISC or CISC impacts performance and power.* We also discuss infrastructure and system challenges, missteps, and software/hardware bugs we encountered. Limitations are addressed in Section 3. Since there are many ways to analyze the raw data, we have released all data at www.cs.wisc.edu/vertical/isa-power-struggles.

Key Findings. The main findings from our study are:

(1) Large performance gaps exist between *implementations*, although average cycle count gaps are predominately $\leq 3\times$.
(2) Instruction count and mix are ISA-independent to first order.
(3) Performance differences are generated by ISA-independent microarchitecture differences.
(4) The energy consumption is again ISA-independent.
(5) ISA differences have implementation implications, but modern microarchitecture techniques render them moot; one ISA is not fundamentally more efficient.

(6) MIPS, ARM, and x86 *implementations* are simply design points optimized for different performance levels.

Implications. Our findings confirm known conventional (or suspected) wisdom and add value by quantification. Our results imply that microarchitectural effects dominate performance, power, and energy impacts. The overall implication of this work is that, although ISA is relevant to power and performance by virtue of support for various specializations (virtualization, accelerators, floating point arithmetic, etc.), the ISA being RISC or CISC is largely irrelevant for today's mature microprocessor design world.

From a broader perspective, our study also points to the role of the ISA for future microprocessors, both for architects and the related fields of systems, compilers, and application development.

Relation to Previous Work. In our previous work [Blem et al. 2013], we analyzed measurements on four platforms—Cortex-A8 (ARM), Cortex-A9 (ARM), Atom (x86), and Sandybridge i7 (x86)—and concluded that ISA being RISC or CISC is irrelevant to performance, power, and energy. In this work, we extend our analysis to three new platforms: Cortex-A15 (ARM), Bobcat (x86), and Loongson2F (MIPS). Through these new platforms, we add an additional ISA (MIPS), an x86 microarchitecture from a non-Intel vendor (AMD's Bobcat), and one of the highest performance ARM implementations (Cortex-A15). Through detailed analysis of our measurement on all seven platforms, we conclude that our main finding still holds true.

Article Organization. Section 2 describes a framework we develop to understand the ISA's impacts on performance, power, and energy. Section 3 describes our overall infrastructure and rationale for the platforms for this study and our limitations, Section 4 discusses our methodology, and Section 5 presents the analysis of our data. Section 6 presents the system and infrastructure challenges faced, and Section 7 concludes the article.

2. FRAMING KEY IMPACTS OF THE ISA

In this section, we present an intellectual framework in which to examine the impact of the ISA—assuming a von Neumann model—on performance, power, and energy. We consider the three key textbook ISA features that are central to the RISC/CISC debate: format, operations, and operands. We do not consider other textbook features, data types and control because they are orthogonal to RISC/CISC design issues, and RISC/CISC approaches are similar. Table I presents the three key ISA features in three columns and their general RISC and CISC characteristics in the first two rows. We then discuss contrasts for each feature and how the choice of RISC or CISC potentially and historically introduced significant tradeoffs in performance and power. In the fourth row, we discuss how modern refinements have led to similarities, thus marginalizing the effect of RISC or CISC on performance and power. Finally, the last row raises empirical questions focused on each feature to quantify or validate this convergence. Overall, our approach is to understand all performance and power differences by using measured metrics to quantify the root cause of differences and whether or not ISA differences contribute. The remainder of this article is centered around these empirical questions framed by the intuition presented as the convergence trends.

3. INFRASTRUCTURE

We now describe our infrastructure and tools. The key take-away is that we pick seven platforms, doing our best to keep them on equal footing, pick representative workloads, and use rigorous methodology and tools for measurement. Readers can skip ahead to Section 4 if uninterested in the details.

Table I. Summary of RISC and CISC Trends

	Format	Operations	Operands
RISC	○ Fixed-length instructions	○ Simple, single-function operations	○ Operands: registers, imm
	○ Relatively simple encoding	○ Single cycle	○ Few addressing modes
	○ ARM: 4B, THUMB (2B, optional)		○ ARM: 16 GPRs
	○ MIPS: 4B		○ MIPS: 32 GPRs
CISC	○ Variable-length instructions	○ Complex, multicycle instructions	○ Operands: memory, registers, imm
	○ Common insts shorter/simpler	○ Transcendentals	○ Many addressing modes
	○ Special insts longer/complex	○ Encryption	○ x86: 8 32b and 6 16b registers
	○ x86: from 1B to 16B long	○ String manipulation	
Historical Contrasts	○ CISC decode latency prevents pipelining	○ Even without μcode in CISC, pipelining hard	○ CISC decoder complexity higher
	○ CISC decoders slower/more area	○ CISC latency may be longer than compiler's RISC equivalent	○ CISC has more per inst work, longer cycles
	○ Code density: RISC < CISC	○ Static code size: RISC > CISC	
Convergence Trends	○ μ-op cache minimizes decoding overheads	○ CISC insts split into RISC-like micro-ops ⇒ optimizations eliminated inefficiency	○ x86 decode optimized for common insts
	○ x86 decode optimized for common insts	○ Modern compilers pick mostly RISC insts ⇒ μ-op counts similar for MIPS, ARM and x86	○ CISC insts split into RISC-like micro-ops ⇒ μ-op latencies similar across ISAs
	○ I-cache minimizes code density impact		○ Number of data cache accesses similar
Empirical Questions	○ How much variance in x86 inst length? Low variance ⇒ common insts optimized	○ Are macro-op counts similar? Similar ⇒ RISC-like on both	○ Number of data accesses similar? Similar ⇒ no data access inefficiencies
	○ Are code densities similar across ISAs? Similar density ⇒ No ISA effect	○ Are complex instructions used by x86 ISA? Few complex ⇒ Compiler picks RISC-like	
	○ What are I-cache miss rates? Low ⇒ caches hide low code densities	○ Are μ-op counts similar? Similar ⇒ CISC split into RISC-like μ-ops	

3.1. Implementation Rationale and Challenges

Choosing implementations presents multiple challenges due to differences in technology (technology node, frequency, high-performance/low-power transistors, etc.), ISA-independent microarchitecture (L2-cache, memory controller, memory size, etc.), design goals (performance, power, energy), and system effects (operating system, compiler, etc.). Finally, it is unfair to compare platforms from vastly different timeframes.

We investigated a wide spectrum of platforms spanning Intel Haswell, Nehalem, Sandybridge, AMD Bobcat, NVIDIA Tegra-2, NVIDIA Tegra-3, and Qualcomm

Table II. Platform Summary

	32/64b x86 ISA			ARMv7 ISA			MIPS
	Sandybridge	Bobcat	Atom	Cortex-A15	Cortex-A9	Cortex-A8	Loongson
Processor	C2700	Zacate E-240	N450	MPCore	OMAP4430	OMAP3530	STLS2F01
Cores	4	2	1	2	2	1	1
Frequency	3.4 GHz	1.5 GHz	1.66 GHz	1.66 GHz	1 GHz	0.6 GHz	0.8 GHz
Width	4-way	2-way	2-way	3-way	2-way	2-way	4-way
Issue	OoO	OoO	In Order	OoO	OoO	In Order	OoO
L1D	32 KB	32 KB	24 KB	32 KB	32 KB	16 KB	64 KB
L1I	32 KB	32 KB	32 KB	32 KB	32 KB	16 KB	64 KB
L2	256 KB/core	512 KB/core	512 KB	1 MB	1 MB/chip	256 KB	512 KB
L3	8 MB/chip	—	—	—	—	—	—
Memory	16 GB	4 GB	1 GB	2 GB	1 GB	256 MB	1 GB
SIMD	AVX	SSE	SSE	NEON	NEON	NEON	—
Area	216 mm^2	—	66 mm^2	—	70 mm^2	60 mm^2	—
Node	32 nm	40 nm	45 nm	32 nm	45 nm	65 nm	90 nm
Platform	Desktop	Dev Board	Dev Board	Dev Board	Pandaboard	Beagleboard	Netbook
Products	Desktop	Netbook	Netbook Lava Xolo	Galaxy S-4	Galaxy S-III Galaxy S-II	iPhone 4, 3GS Motorola Droid	Lemote Yeelong

Snapdragon. However, we did not find implementations that met all of our criteria: same technology node across the different ISAs, identical or similar microarchitecture, development board that supported necessary measurements, a well-supported operating system, and similar I/O and memory subsystems. We ultimately picked the Cortex-A8 (ARM), Cortex-A9 (ARM), Cortex-A15 (ARM), Atom (x86), Bobcat (x86), Loongson (MIPS), and i7 (x86) Sandybridge processor. We choose A8, A9, Atom, and Bobcat because they include processors with similar microarchitectural features like issue-width, caches, and main-memory and are from similar technology nodes, as described in Tables II and VIII. They are all relevant commercially, as shown by the last row in Table II. For a high-performance x86 implementation, we use an Intel i7 Sandybridge processor; it is significantly more power-efficient than any 45nm offering, including Nehalem. Intel Haswell is implemented at 22nm, and the technology advantages made it a less desirable candidate for our goal of studying architecture and microarchitecture issues. For a high-performance ARM implementation, we use the A15 processor; it is a significant upgrade to the A9 microarchitecture and aims to maximize performance. We chose to include the Loongson processor as representative of a true RISC ISA (MIPS) implementation. Importantly, these choices provided usable software platforms in terms of operating system, cross-compilation, and driver support. Overall, our choice of platforms provides a reasonably equal footing, and we perform detailed analysis to isolate out microarchitecture and technology effects. We present system details of our platforms for context, although the focus of our work is the processor core.

A key challenge in running real workloads was the relatively small memory (512MB) on the Cortex-A8 Beagleboard. Although representative of the typical target (e.g., iPhone 4 has 512MB RAM), it presents a challenge for workloads like SPECCPU2006; execution times are dominated by swapping and OS overheads, making the core irrelevant. Section 3.3 describes how we handled this. In the remainder of this section, we discuss the platforms, applications, and tools for this study in detail.

3.2. Implementation Platforms

Hardware Platform. We consider three ARM, one MIPS, and three x86 ISA implementations as described in Table II.

Intent. Keep nonprocessor features as similar as possible.

Table III. Benchmark Summary

Domain	Benchmarks	Notes
Mobile client	CoreMark	Set to 4000 iterations
	WebKit	Similar to BBench
Desktop	SPECCPU2006	10 INT, 10 FP, test inputs
Server	lighttpd	Represents web-serving
	CLucene	Represents web-indexing
	Database kernels	Represents data-streaming and data-analytics

Operating System. Across all platforms except A15, we run the same stable Linux 2.6 LTS kernel with some minor board-specific patches to obtain accurate results when using the performance counter subsystem. We run Linux 3.8 on A15 because we encountered many technical challenges while trying to backport performance counters support to Linux 2.6.

Intent. Keep OS effects as similar as possible across platforms.

Compiler. Our toolchain is based on a validated `gcc` 4.4-based cross-compiler configuration. We intentionally chose `gcc` so that we can use the same front-end to generate all binaries. All target independent optimizations are enabled (O3) and machine-specific tuning is disabled in order to maintain the same set of binaries for all platforms of the same ISA. Disabling machine-specific tuning is justified since any improvement in performance and/or energy due to machine-specific tuning is, by definition, a microarchitecture artifact and is not related to the ISA being RISC or CISC. All binaries are 32-bit since 64-bit ARM platforms are still under development. For ARM, we disable THUMB instructions for a more RISC-like ISA. None of the benchmarks includes SIMD code, and although we allow autovectorization, very few SIMD instructions are generated for either ARM or x86 architectures. As for Loongson, although its instruction set supports SIMD instructions, they are not part of the MIPS III ISA. The `gcc` MIPS compiler that we use does not generate any Loongson-specific instructions. Floating point is done natively on the SSE units on x86 implementations, NEON units on ARM implementations, and the floating-point unit on Loongson. Vendor compilers may produce better code for a platform, but we use `gcc` to eliminate compiler influence. As seen in Table XV of Appendix I, static code size is within 8% and average instruction lengths are within 4% using `gcc` and `icc` for SPEC INT, so we expect that compiler does not make a significant difference.

Intent. Hold compiler effects constant across platforms.

3.3. Applications

Since all ISAs studied in this work are touted as candidates for mobile clients, desktops, and servers, we consider a suite of workloads that span these. We use prior workload studies to guide our choice, and, where appropriate, we pick equivalent workloads that can run on our evaluation platforms. A detailed description follows and is summarized in Table III. All workloads are single-threaded to ensure our single-core focus (see Section 3.5). Next, we discuss each suite in turn.

Mobile Client. This category presented challenges because mobile client chipsets typically include several accelerators and careful analysis is required to determine the typical workload executed on the programmable general-purpose core. We used CoreMark (www.coremark.org), widely used in industry white-papers, and two WebKit regression tests informed by the BBench study [Gutierrez et al. 2011]. BBench, a recently proposed smartphone benchmark suite, is a "web-page rendering benchmark comprising 11 of the most popular sites on the internet today" [Gutierrez et al. 2011].

Table IV. Infrastructure Limitations

	Limitation	Implications
Cores	Multicore effects: coherence, locking...	Second-order for core design
	No platform uniformity across ISAs	Best effort
	No platform diversity within ISAs	Best effort
	Design teams are different	μarch effect, not ISA
Domain	Ultra-low-power microcontrollers	Out of scope
	Server style platforms	See server benchmarks
	Why SPEC on mobile platforms?	Tracks emerging uses
	Why not SPEC JBB or TPC-C?	CloudSuite more relevant
Tools	Proprietary compilers are optimized	`gcc` optimizations uniform
	Arch. specific compiler tuning	<10%
	No direct decoder power measure	Results show second-order
	Power includes noncore factors	4%–17%
	Performance counters may have errors	Validated use (Table V)
	Limited performance counters	Only cycle and instruction counters on Loongson
	Simulations have errors	Validated use (Table V)
Scaling	Memory rate effects cycles nonlinearly	Second-order
	Vmin limit effects frequency scaling	Second-order
	ITRS scaling numbers are not exact	Best effort; extant nodes

To avoid web-browser differences across the platforms, we use the cross-platform WebKit with two of its built-in tests that mimic real-world HTML layout and performance scenarios for our study.[2] We did not run the WebKit tests on Loongson due to the unavailability of a machine-optimized MIPS binary. Since we use optimized binaries of this benchmark on other platforms, reporting WebKit numbers using an unoptimized binary on Loongson would constitute an unfair comparison.

Desktop. We use the SPECCPU2006 suite (www.spec.org) as representative of desktop workloads. SPECCPU2006 is a well understood standard desktop benchmark that provides insights into core behavior. Due to the large memory footprint of the train and reference inputs, we found that, for many benchmarks, the memory-constrained Cortex-A8 ran of memory, and execution was dominated by system effects. Hence, we report results using the test inputs, which fit in the Cortex-A8's memory footprint for 10 of 12 INT and 10 of 17 FP benchmarks.

Server. We chose server workloads informed by the recently proposed CloudSuite workloads [Ferdman et al. 2012]. Their study characterizes server/cloud workloads into data analytics, data streaming, media streaming, software testing, web search, and web serving. The actual software implementations they provide are targeted for large memory-footprint machines, and their intent is to benchmark the *entire system and server cluster*. This is unsuitable for our study since we want to isolate processor effects. Hence, we pick implementations with small memory footprints and single-node behavior. To represent data-streaming and data-analytics, we use three database kernels commonly used in database evaluation work [Rao and Ross 2000; Kim et al. 2009] that capture the core computation in Bayes classification and datastore.[3] To represent web search, we use `CLucene` (clucene.sourceforge.net), an efficient,

[2] Specifically `coreLayout` and `DOMPerformance`.

[3] CloudSuite uses Hadoop+Mahout plus additional software infrastructure, ultimately running Bayes classification and data store; we feel this kernel approach is better suited for our study while capturing the domain's essence.

cross-platform indexing implementation similar to CloudSuite's Nutch. To represent web-serving (CloudSuite uses Apache), we use the `lighttpd` server (www.lighttpd.net), which is designed for "security, speed, compliance, and flexibility." We do not evaluate the media-streaming CloudSuite benchmark because it primarily stresses the I/O subsystem. CloudSuite's Software Testing benchmark is a batch coarse-grained parallel symbolic execution application; for our purposes, the SPEC suite's Perl parser, combinational optimization, and linear programming benchmarks are similar.

3.4. Tools

The four main tools we use in our work are described here, and Table V describes how we use them.

Native execution time and microarchitectural events. We use wall-clock time and performance-counter-based clock-cycle measurements to determine execution time of programs. We also use performance counters to understand microarchitecture influences on the execution time. Each of the processors has different counters available, and we examined them to find comparable measures. Ultimately, three counters explain much of the program behavior: branch misprediction rate, Level-1 data cache miss rate, and Level-1 instruction cache miss rate (all measured as misses per kilo-instructions). We use the `perf` tool for performance counter measurement.

Power. For power measurements, we connect a Wattsup (www.wattsupmeters.com) meter to the board/desktop/laptop power supply. This gives us system power. We run the benchmark repeatedly to find consistent average power as explained in Table V. We use a control run to determine the board power alone when the processor is halted and subtract away this board power to determine chip power. Some recent power studies [Esmaeilzadeh et al. 2011; Isci and Martonosi 2003; Bircher and John 2008] accurately isolate the processor power alone by measuring the current supply line of the processor. This is not possible for the SoC-based ARM development boards, and hence we determine and then subtract out the board power. This methodology allows us to eliminate the main memory and I/O power and examine only processor power.[4] On the Loongson netbook, we tweaked the software DVFS governor to put the processor into low-power mode in order to compute the idle power. System power was computed by removing the netbook battery and connecting its power supply to wall socket via the WattsUp meter. To remove LCD power, we wait for the LCD to enter standby state before taking power measurements. We validated our strategy for the i7 system using the exposed energy counters (the only platform we consider that includes isolated power measures). Across all three benchmark suites, our WattsUp methodology compared to the processor energy counter reports ranged from 4% to 17% less, averaging 12%. Our approach tends to underestimate core power, so our results for power and energy are optimistic. We saw average power of 840mW, 1.1W, 2.4W, 21.3W, 4.5W, 5.5W, 2.7W for A8, A9, Atom, i7, Bobcat, A15, and Loongson, respectively, and these fall within the typical vendor-reported power numbers.

Technology scaling and projections. Since the i7 processor and Cortex-A8 are based on 32nm and 65nm technology nodes, respectively, we use technology node characteristics from the 2007 ITRS tables to normalize to the 45nm technology node (A9 and Atom are 45nm) in two results where we factor out technology; we do not account for device type (LOP, HP, LSTP). We normalize Bobcat (40nm) to 45nm using the weighted average of the 32nm and 45nm numbers. We normalize Loongson (90nm) to 45nm by equating the slope from 90nm to 65nm to the slope that we see from 65nm to 45nm. For

[4] According to the WattsUp meter vendor, the meter's wattage accuracy is within +/− 1.5%.

Table V. Methodology Summary

	(a) Native Execution on Real Hardware
Measures	Methodology
Execution time Cycle counts	○ Approach: Use `perf` tool to sample cycle performance counters; sampling avoids potential counter overflow. Run one benchmark at a time to make sure that only a single core is exercised in the processors. ○ Analysis: 5–20 trials (dependent on variance and benchmark runtime); report minimum from trials that complete normally. ○ Validation: Compare against wall clock time.
Inst. count (ARM)	○ Approach: Use `perf` tool to collect macro-ops from performance counters ○ Analysis: At least three trials; report minimum from trials that complete normally. ○ Validation: ARM Performance counters within 10% of gem5 ARM simulation. Table XIII elaborates on challenges.
Inst. count (MIPS)	○ Approach: Use `perf` tool to collect macro-ops from performance counters ○ Analysis: At least three trials; report minimum from trials that complete normally. ○ Validation: No validation performed.
Inst. count (x86)	○ Approach: Use `perf` to collect macro-ops and micro-ops from performance counters. ○ Analysis: At least three trials; report minimum from trials that complete normally. ○ Validation: Counters within 2% of DynamoRIO trace count (macro-ops only). Table XIII elaborates on challenges.
Inst. mix (Coarse)	○ Approach: SIMD + FP + load/store performance counters. No mix data collected on MIPS platform. Table XIII elaborates on challenges.
Inst. length (x86)	○ Approach: Wrote Pin tool to find length of each instruction and keep running average.
Microarch events	○ Approach: Branch mispredictions, cache misses, and other uarch events measured using `perf` performance counters. uarch events not collected on MIPS platform. Table XIII elaborates on challenges. ○ Analysis: At least three trials; additional if a particular counter varies by >5%. Report minimum from normal trials.
Full system power	○ Set-up: Use Wattsup meter connected to board, desktop or netbook (no network connection, peripherals on separate supply, kernel DVFS disabled, cpuidle support disabled, cores at peak frequency, single-user mode). ○ Approach: Run benchmarks in loop to guarantee 3 minutes of samples (180 samples at maximum sampling rate). ○ Analysis: If outliers occur, rerun experiment; present average power across run without outliers.
Board power	○ Set-up: Use Wattsup meter connected to board, desktop, or netbook (no network connection, peripherals on separate supply, kernel DVFS disabled, cpuidle support disabled, cores at peak frequency, single-user mode). ○ Approach: Run with kernel power saving enabled; force to lowest frequency. Issue halt; report power when it stabilizes. ○ Analysis: Report minimum observed power.
Processor power	○ Approach: Subtracting above two gives processor power. ○ Validation: Compare core power against energy performance counters and/or reported TDP and power draw.
	(b) Emulated Execution
Measures	Methodology
Inst. mix (Detailed)	○ Approach (ARM): Use gem5 instruction trace and analyze using python script. ○ Approach (MIPS): Not collected on MIPS platform. ○ Approach (x86): Use DynamoRIO instruction trace and analyze using python script. ○ Validation: Compare against coarse mix from SIMD + FP + load/store performance counters.
ILP	○ Approach: Pin-based MICA tool that reports ILP with window size 32, 64, 128, 256.

our 45nm projections, the A8's power is scaled by 0.8×, i7's by 1.3×, Bobcat's by 1.1×, A15's by 1.3×, and Loongson's by 0.6×. In some results, we scale frequency to 1GHz, accounting for DVFS impact on voltage using the mappings disclosed for Intel SCC [Baron 2010]. When frequency scaling, we assume that 20% of the i7's power is static and does not scale with frequency; all other cores are assumed to have negligible static power. When frequency scaling, A8's power is scaled by 1.2×, Atom's by 0.8×, i7's by 0.6×, Bobcat's by 0.8×, A15's by 0.8×, and Loongson's by 1.2×. We acknowledge that this scaling introduces some error to our technology-scaled power comparison, but we feel it is a reasonable strategy and does not affect our primary findings (see Table IV).

Emulated instruction mix measurement. For the x86 ISA, we use DynamoRIO [Bruening et al. 2003] to measure instruction mix. For the ARM ISA, we leverage the gem5 [Binkert et al. 2011] simulator's functional emulator to derive instruction mixes (no ARM binary emulation available). Due to tool limitations, we could not gather any instruction mix data for Loongson. Our server and mobile-client benchmarks use many system calls that do not work in the gem5 functional mode. We do not present detailed instruction-mix analysis for these, but instead present a high-level mix determined from performance counters. We use the MICA tool to find the available ILP [Hoste and Eeckhout 2007].

3.5. Limitations or Concerns

Our study's limitations are classified into core diversity, domain, tool, and scaling effects. The full list appears in Table IV, and details are discussed here. Throughout our work, we focus on what we believe to be the first-order effects for performance, power, and energy and feel our analysis and methodology is rigorous. Other more detailed methods may exist, and we have made the data publicly available at www.cs.wisc.edu/vertical/isa-power-struggles to allow interested readers to pursue their own detailed analysis.

Cores. We considered seven platforms: three x86, three ARM, and one MIPS. A perfect study would include platforms at several performance levels with matched frequency, branch predictors, other microarchitectural features, and memory systems. Furthermore, a pure RISC versus CISC study would use "true" RISC and CISC cores, whereas ARM and x86's ISA tweaks represent the current state of the art. Only Loongson may claim to implement a true RISC ISA. Our selections reflect the available and well-supported implementations.

Domain. We picked a representative set of workloads that captures a significant subset of modern workloads. We do not make broad domain-specific arguments since that requires truly representative inputs and IO subsystem control for the mobile and server domains. Our study focuses on single-core and thus intentionally avoids multicore system issues (e.g., consistency models, coherence, virtualization, etc.).

Measurement and Tool Limitations. Our measurements are primarily on real hardware and therefore include real-world errors. We execute multiple runs and take a rigorous approach, as detailed in Table V. Eliminating all errors is impractical, and our final result trends are consistent and intuitive.

Analysis. We have presented our analysis of this rich dataset. We will release the data to allow interested readers to pursue their own detailed analysis.

4. METHODOLOGY

In this section, we describe how we use our tools and the overall flow of our analysis. Section 5 presents our data and analysis. Table V describes how we employ the aforementioned tools and obtain the measures we are interested in, namely, execution time, execution cycles, instruction mix, microarchitecture events, power, and energy.

Our overall approach is to understand all performance and power *differences* and to use the measured metrics to quantify the root cause of differences and whether or not ISA differences contribute, thus answering empirical questions from Section 2. Unless otherwise explicitly stated, all data are measured on real hardware. The flow of the next section is outlined next.

4.1. Performance Analysis Flow

Step 1: Present execution time for each benchmark
Step 2: Normalize frequency's impact using cycle counts
Step 3: To understand differences in cycle count and the influence of the ISA, present the dynamic instruction count measures, measured in both macro-ops and micro-ops
Step 4: Use instruction mix, code binary size, and average dynamic instruction length to understand ISA's influence
Step 5: To understand performance differences not attributable to ISA, look at detailed microarchitecture events
Step 6: Attribute performance gaps to frequency, ISA, or ISA-independent microarchitecture features; qualitatively reason about whether the ISA forces microarchitecture features

4.2. Power and Energy Analysis Flow

Step 1: Present per-benchmark raw power measurements
Step 2: To factor out the impact of technology, present technology-independent power by scaling all processors to 45nm and normalizing the frequency to 1GHz
Step 3: To understand the interplay between power and performance, examine raw energy
Step 4: Qualitatively reason about the ISA influence on microarchitecture in terms of energy

4.3. Tradeoff Analysis Flow

Step 1: Combining the performance and power measures, compare the processor implementations using Pareto frontiers
Step 2: Compare processor implementations using Energy Performance Pareto frontiers

5. MEASURED DATA ANALYSIS AND FINDINGS

We now present our measurements and analysis of performance, power, energy, and the tradeoffs among them.

We present our data for all seven platforms, often comparing A8 to Atom (both dual-issue in-order) and Loongson, A9, A15, and Bobcat to i7 (all out-of-order). For each step, we present the average measured data, average in-order and OoO ratios if applicable, and then our main findings. We separate the set of bars and ratios for in-order cores from those for out-of-order cores with a gap in the relevant figures and corresponding tables. For detailed (per-benchmark) measurement data, refer to Appendix I.

In Figures 2, 3, 4(a), 8, 9, and 10 and the corresponding tables, the averages including outliers are shown in parentheses. The averages excluding outliers are depicted by the bar heights in the figures and by the numbers outside parentheses in the tables. We categorize a certain benchmark as an outlier if we are not able to run it on one of the platforms or if its instruction count is unexpectedly high or low. In our analysis, we show that these abnormalities in instruction count are not due to the ISA. In the figures and tables mentioned, we categorize the following benchmarks as outliers:

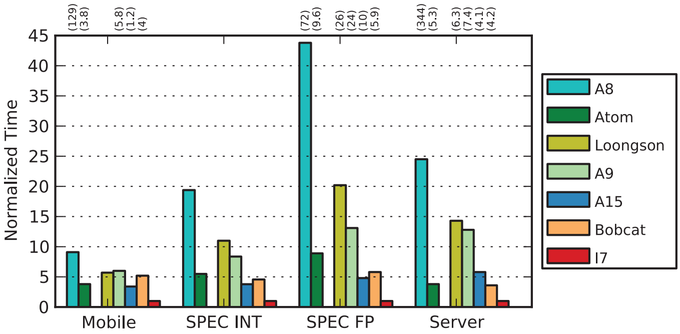


Fig. 2. Execution time normalized to i7.

`webkit` (MOBILE), `tonto` (SPECFP), `cactusADM` (SPECFP), `bwaves` (SPECFP), `clucene` (SERVER), and `lighttpd` (SERVER).

5.1. Performance Analysis

Step 1: *Execution Time Comparison*

Data: Figure 2 shows execution time normalized to i7. Average ratios are in the included table. Per- benchmark data are shown in Figure 15 of Appendix I.

Ratio	Mobile	SPEC INT	SPEC FP	Server
A8/Atom	2.4 (34)	3.5	4.9 (7.4)	6.5 (64.7)
Loongson/i7	5.7	11	20.2 (26.2)	14.3 (6.3)
A9/i7	6 (5.8)	8.4	13.1 (23.8)	12.8 (7.4)
A15/i7	3.4 (1.2)	3.8	4.8 (10.1)	5.8 (4.1)
Bobcat/i7	5.2 (4)	4.6	5.8 (5.9)	3.6 (4.2)

Outliers: See Tables IX, X, XI, and XII. Where outliers are listed, they are in this set.

Outlier 1: A8 performs particularly poorly on `webkit` tests and `lighttpd`.

Outlier 2: Loongson performs poorly on `hmmer` (specINT), database, and some SPECFP benchmarks.

Outlier 3: A9 performs poorly on database kernels and some SPEC FP benchmarks.

Outlier 4: A15 performs particularly well on `webkit` tests (45× faster than A9[5]) and poorly on some SPEC FP benchmarks.

Outlier 5: Bobcat performs poorly on some SPEC FP benchmarks.

Finding P1: Large performance gaps are platform and benchmark dependent: A8/Atom gaps range from 2× to 152×, Loongson/i7 performance gaps range from 1× to 56×, A9/i7 performance gaps range from 4× to 102×, A15/i7 performance gaps range from 0.06× to 50×, and Bobcat/i7 performance gaps range from 3× to 11×.

Key Finding 1: Large performance gaps exist across the seven platforms studied, as expected, since frequency ranges from 600MHz to 3.4GHz and microarchitectures are very different.

[5] Such a large performance gap cannot be attributed to the microarchitecture difference between A15 and A9, despite the fact that A15 is a significant upgrade to A9. Most likely, this gap is due to system software issues that we haven't been able to isolate.

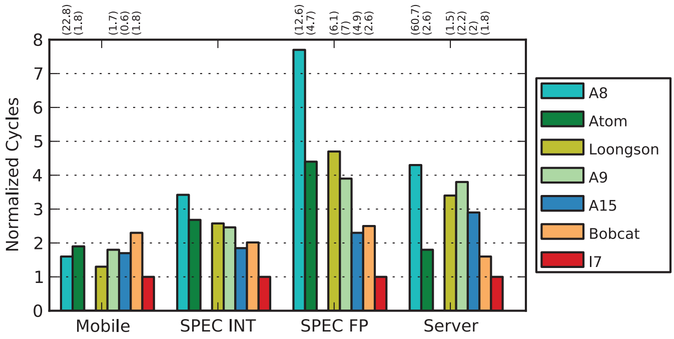


Fig. 3. Cycle count normalized to i7.

Step 2: *Cycle-Count Comparison*

Data: Figure 3 shows cycle counts normalized to i7. Average ratios are in the included table. Per- benchmark data are in Figures 7(a), 7(b), and 7(c).

Outliers: After normalizing the performance gaps for frequency, we still see the same outliers as in Step 1.

Finding P2: Despite being from different ISAs, mean cycle count gap of out-of-order implementations Loongson, A9, A15, and Bobcat with respect to i7, across all suites, is predominantly less than 3× (not considering outliers).

Ratio	Mobile	SPEC INT	SPEC FP	Server
A8/Atom	0.9 (12.5)	1.3	1.8 (2.7)	2.4 (23.4)
Loongson/i7	1.3	2.6	4.7 (6.2)	3.4 (1.5)
A9/i7	1.8 (1.7)	2.5	3.9 (7)	3.8 (2.2)
A15/i7	1.7 (0.6)	1.9	2.3 (4.9)	2.9 (2)
Bobcat/i7	2.3 (1.8)	2	2.5 (2.6)	1.6 (1.8)

Finding P3: Per suite cycle count gap between in-order ARM and x86 implementations (A8 and Atom) is predominantly less than 2× (not considering outliers).

Key Finding 2: Performance gaps, when normalized to cycle counts, are predominantly less than 3× when comparing in-order cores to each other and out-of-order cores to each other.

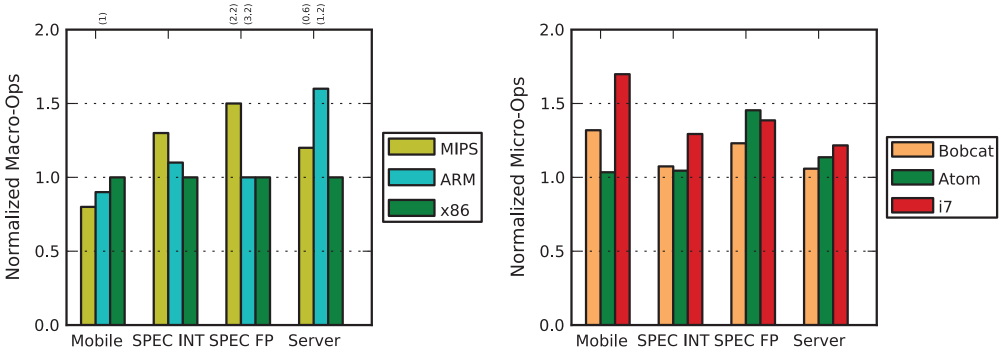
Step 3: *Instruction Count Comparison*

Data: Figure 4(a) shows dynamic instruction (macro) counts for Loongson, A8, and Atom normalized to Atom x86 macro-instructions. We choose A8 and Atom as representatives of their respective ISAs since instruction counts should be similar across cores of the same ISA. Per-benchmark data are shown in Figure 14(a) of Appendix I.

Data: Table VI shows the geometric mean CPI for all implementations across all benchmark suites. Per-benchmark data are shown in Table XVI of Appendix I.

Data: Figure 4(b) shows dynamic micro-op counts for Bobcat, Atom, and i7 normalized to Atom macro-instructions.[6] Per-benchmark data are shown in Figure 14(b) of Appendix I.

[6] For i7, we use issued micro-ops instead of retired micro-ops; we found that, on average, this does not impact the micro-op-to-macro-op ratio.



(a) Macro-Ops

(b) Micro-Ops

Fig. 4. Instructions normalized to Atom macro-ops.

MIPS Outliers:

Outlier 1: For lighttpd, Loongson executes about 8× fewer instructions than x86 and ARM machines. We were not able to isolate the reason behind this due to insufficient analysis tools (binary instrumentation tools for MIPS do not exist, and the Loongson core has limited performance counters).

Outlier 2: Typically, for SPEC FP benchmarks, Loongson executes about 2× more instructions than x86 machines. We suspect that a more efficient math library on x86 could account for this difference. Again, we could not validate this quantitatively due to lack of analysis tools.

ARM Outliers:

Outlier 3: For wkperf and lighttpd, A8 executes more than twice as many instructions as A9.[7] We report A9 instruction counts for these two benchmarks.

Outlier 4: All ARM outliers in SPEC FP are due to transcendental FP operations supported only by x86.

x86 Outliers:

Outlier 5: For Clucene, x86 machines execute 1.7× more instructions than ARM machines and 2.3× more instructions than Loongson. These appear to be pathological cases of x86 code generation inefficiencies.

Outlier 6: For cactusADM, Atom executes 2.7× more micro-ops than macro-ops.

Outlier 7: For webkit, i7 executes about 2× more micro-ops than atom.

Finding P4: Instruction count is similar across ISAs (excluding system and library overheads). This implies that gcc often picks RISC-like instructions from the x86 ISA.

Finding P5: x86 micro-op to macro-op ratio is often less than 1.3×, again suggesting gcc picks the RISC-like instructions.

Key Finding 3: Despite similar instruction counts across ISAs, CPI can be *less* on x86 implementations across all suites (as shown in Table VI). This finding disproves prior

[7] A8 spins for IO, event-loops, and timeouts.

Table VI. Geometric Mean CPI a cross All Benchmark Suites

ISA	ARM	x86	MIPS	ARM	ARM	x86	x86
Implementation	A8	Atom	Loongson	A9	A15	Bobcat	i7
CPI	2.5	1.9	1.5	1.6	1.2	1.3	0.7

Table VII. Instruction Size Summary

		(a) Binary Size (MB)			(b) Instruction Length (B)		
		MIPS	ARM	x86	MIPS	ARM	x86
Mobile	Minimum	–	0.02	0.02	4.0	4.0	2.4
	Average	0.55	0.95	0.87	4.0	4.0	3.3
	Maximum	–	1.30	1.42	4.0	4.0	3.7
Desktop INT	Minimum	0.61	0.53	0.65	4.0	4.0	2.7
	Average	1.58	1.47	1.46	4.0	4.0	3.1
	Maximum	4.35	3.88	4.05	4.0	4.0	3.5
Desktop FP	Minimum	0.76	0.66	0.74	4.0	4.0	2.6
	Average	1.81	1.70	1.73	4.0	4.0	3.4
	Maximum	5.21	4.75	5.24	4.0	4.0	6.4
Server	Minimum	0.16	0.12	0.18	4.0	4.0	2.5
	Average	0.51	0.39	0.59	4.0	4.0	3.2
	Maximum	0.84	0.47	1.00	4.0	4.0	3.7

belief that CISC implementations should have a higher CPI than RISC implementations (due to the complex instructions in CISC). Microarchitecture is the dominant factor that affects performance, not the ISA.

Step 4: *Instruction Format and Mix*

Data: Table VII(a) shows average MIPS, ARM, and x86 static binary sizes,[8] measuring only the binary's code sections. Per-benchmark data are shown in Table XV(a) of Appendix I.

Data: Table VII(b) shows average dynamic MIPS, ARM, and x86 instruction lengths. Per-benchmark data are shown in Table XV(b) of Appendix I.

Finding P6: Average ARM and x86 binary sizes are similar for SPEC INT, SPEC FP, and Mobile workloads, thus suggesting similar code densities.

Finding P7: Executed x86 instructions are on average up to 25% shorter than ARM and MIPS instructions: short, simple x86 instructions are typical.

Finding P8: x86 FP benchmarks, which tend to have more complex instructions, have instructions with longer encodings (e.g., `cactusADM` with 6.4 Bytes/inst on average).

Finding P9: MIPS binary sizes are similar to ARM and x86 for specINT and specFP suites. For mobile and server suites, we do not have enough control over the software stack for some benchmarks, and hence it would be unfair to compare their instruction counts to ARM and x86.

Data: Figure 5 shows average coarse-grained ARM and x86 instruction mixes for each benchmark suite.[9]

Data: We do not have instruction mix data for MIPS because we were unable to gather performance counter data beyond cycle and instruction counts (see Section 6).

[8] On Loongson, binary size data are robust only for specINT and specFP (explained in finding P9). We measure data only for one benchmark from the mobile suite (coremark).

[9] x86 instructions with memory operands are cracked into a memory operation and the original operation.

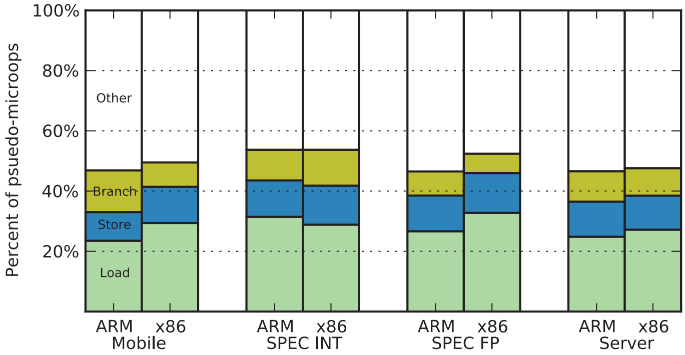


Fig. 5. Instruction mix (performance counters).

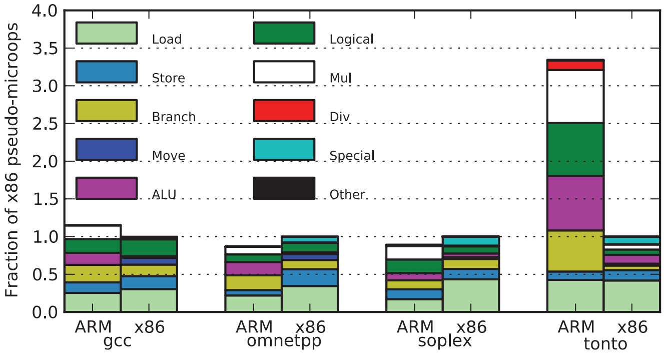


Fig. 6. Selected instruction counts (emulated).

Data: Figure 6 shows fine-grained ARM and x86 instruction mixes normalized to x86 for a subset of SPEC benchmarks.

Finding P10: Fraction of loads and stores similar across ISA for all suites (we do not have these data for MIPS), suggesting that the ISA does not lead to significant differences in data accesses.

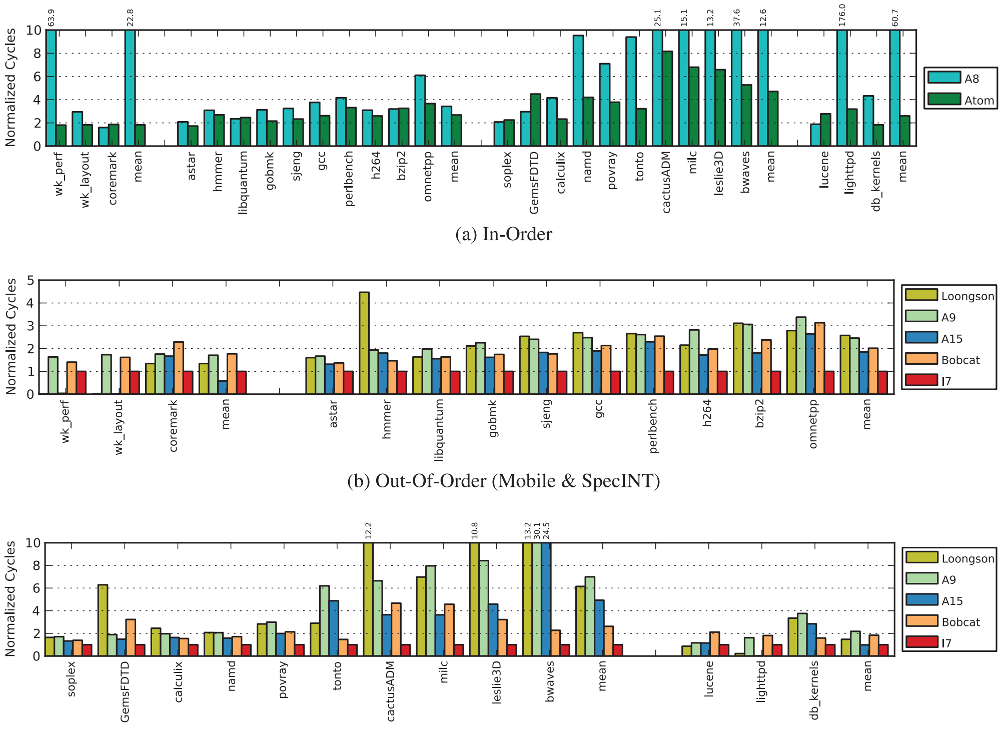
Finding P11: Large instruction counts for ARM are due to absence of FP instructions like `fsincon`, `fyl2xpl`, and the like (e.g., `tonto` in Figure 6's many special x86 instructions correspond to ALU/logical/multiply ARM instructions).

Key Finding 4: Combining the instruction count and mix findings, we conclude that ISA effects are indistinguishable between x86 and ARM implementations. Due to infrastructure limitations, we do not have enough data to make the same claim for the MIPS platform. However, we suspect that instruction count differences on MIPS platform are due to system software issues and not due to the ISA.

Step 5: *Microarchitecture*

Data: Figure 7 shows per-benchmark cycle counts for more detailed analysis where performance gaps are large.

Data: Figures 16, 17, and 18 in Appendix I show per-benchmark branch mispredictions, L1 data, and instruction cache misses per 1,000 ARM instructions for ARM and x86 platforms.



(a) In-Order

(b) Out-Of-Order (Mobile & SpecINT)

(c) Out-Of-Order (SpecFP & Server)

Fig. 7. Per-benchmark core cycle counts normalized to i7.

Table VIII. Processor Microarchitecture Features

(a) In-Order Cores

	Pipeline Depth	Issue Width	Threads	ALU/FP Units	Br. Pred. BTB Entries
A8	13	2	1	2/2 + NEON	512
Atom	16 + 2	2	2	2/2 + IMul	128

(b) Out-of-Order Cores

	Issue		ROB		Entries for		
	width	Threads	Size	LD/ST	Rename	Scheduler	BTB
Loongson	4	1	64	16	96	32	16
A9	2	1	–	–/4	56	20	512
A15	3	1	60	16/16	112	40	256
Bobcat	2	1	–	–	–	–	–
i7	4(6)	2	64/36	160	168	54	8K–16K

Data: For Loongson (MIPS) platform, we do not present detailed analysis in this section due to the absence of event counts like branch and cache misses. Toolchain problems prevented us from being able to collect these data.

Data: Table VIII compares the primary structures of the in-order and out-of-order cores. These details are from six Microprocessor Report articles and Loongson manual.[10] The

[10] "Cortex-A8 High speed, low power" (Nov 2005), "More applications for OMAP4" (Nov 2009), "Sandybridge spans generations" (Sept 2010), "Intel's Tiny Atom" (April 2008), "Cortex A-15 Eagle Flies the Coop" (Nov

Table IX. Detailed Analysis for Benchmarks with A8 to Atom Cycle Count Gap Greater than 3×

Benchmark	Gap	Analysis
tonto	2.9	Instructions: 4.7× more for ARM than x86
cactusADM	3.1	Instructions: 2.8× more for ARM than x86
bwaves	7.1	Instructions: 17.5× more for ARM than x86

Table X. Detailed Analysis for Benchmarks with A9 to i7 Cycle Count Gap Greater than 3×

Benchmark	Gap	Analysis (Issue width difference explains up to 2× Gap)
omnetpp	3.4	Branch MPKI is 59 for A9 versus only 2.0 for i7. I-Cache MPKI is 33 for A9 versus only 2.2 for i7.
db_kernels	3.8	1.6× more instructions, 5× more branch MPKI for A9 than i7
tonto	6.2	Instructions: 4× more for ARM than x86
cactusADM	6.6	Instructions: 2.8× more for ARM than x86
milc	8.0	Branch MPKI is 61× more for A9 than i7.
leslie3D	8.4	Branch MPKI is 7× more for A9 than i7. Data cache MPKI is 55 for A9, compared to only 30 for the i7.
bwaves	30	Instructions: 17.5× more for ARM than x86

Table XI. Detailed Analysis for Benchmarks with A15 to i7 Cycle Count Gap Greater than 3×

Benchmark	Gap	Analysis (Issue width difference explains up to 1.3× Gap)
tonto	4.9	Instructions: 4× more for ARM than x86
cactusADM	3.7	Instructions: 2.8× more for ARM than x86
milc	3.6	Branch MPKI is 30× more for A15 than i7
leslie3D	4.6	Branch MPKI is 6× more for A15 than i7. Data Cache MPKI is 0.1 for A15 in comparison to 30 for i7, which explains the improved performance on A15 compared to A9. It also shows that i7's microarchitecture hides the data cache misses more effectively than does A15 (and A9)
bwaves	25	Instructions: 17.5× more for ARM than x86

A9 numbers are estimates derived from publicly disclosed information on A15 and A9/A15 comparisons.

Finding P12: When comparing two processors, the issue width ratio of K explains cycle count gap up to K, assuming sufficient ILP, a sufficient instruction window, and a well-balanced processor pipeline.[11] The performance gaps not explained by issue width can be attributed to other microarchitectural event counts (e.g., branch misses on A9, A15, and Bobcat are more common than on i7). These differences are not because of the ISA, but rather due to microarchitectural design choices (e.g., A9 and A15 have 512 and 256 BTB entries, respectively, versus i7's 16K entries[12]).

Finding P13: Per benchmark, we can attribute the largest gaps in performance between i7 and other out-of-order processors (and between Atom & A8) to specific microachitectural events. In the interest of space, we present detailed performance analysis for benchmarks with gaps greater than 3× in Tables IX, X, XI, and XII. Building on Table XII, we present more elaborate analysis of Bobcat versus i7 performance. We also present performance analysis of Loongson's performance with respect to i7. We do not present elaborate analysis for other tables since they sufficiently explain the

2010), "AMD's Bobcat snarls at Atom" (August 2010), Loongson User Manual: http://dev.lemote.com/files/resource/documents/Loongson/ls2f/Loongson2FUserGuide.pdf.

[11] We use MICA to confirm that our benchmarks all have limit ILP >4 [Hoste and Eeckhout 2007].

[12] Bobcat has not released its BTB size, but its mean branch MPKI overall benchmarks is higher than i7.

Table XII. Detailed Analysis for Benchmarks with Bobcat to i7 Cycle Count Gap Greater than 3×

Benchmark	Gap	Analysis (Issue width difference explains up to 2× Gap)
omnetpp	3.1	Branch MPKI is 14× more and I-Cache MPKI is 11× more for Bobcat than i7.
GemsFDTD	3.2	Branch MPKI is 5× more and I-Cache MPKI is 6× more for Bobcat than i7.
cactusADM	4.7	Branch MPKI and I-Cache MPKI is 3× more for Bobcat than i7.
milc	4.6	Branch MPKI is 15× more for Bobcat than i7.

performance gaps. Furthermore, we present detailed analysis of bwaves performance, comparing ARM and x86 and explaining their large cycle count gap.

Analysis of Bobcat versus i7 Performance:

Table XII analyzes performance gaps between Bobcat and i7. We see that much of the performance gap (beyond first-order issue width effect) can be attributed to Branch and I-cache misses. In particular, we notice a discrepency where a branch MPKI gap of 3× on cactusADM has similar impact on performance as a branch MPKI gap of 15× on milc. This difference is not beyond reason because some misses can be more expensive for one workload compared to another. Detailed analysis of this behavior is tangential to the goal of this work since both Bobcat and i7 are x86 ISA implementations.

Analysis of Loongson vs i7 Performance:

The mean performance gap between Loongson and i7 is 3.4×. Apart from two benchmarks where the gap is 1.6×, all other benchmarks perform more than 2× worse on Loongson compared to i7 despite both platforms having the same issue width. These differences may be attributed to smaller load/store queue, rename registers, and BTB size on Loongson, but we do not have detailed data on this platform to support this argument.

Analysis of bwaves performance on ARM vs x86:

The bwaves benchmark performed significantly worse (up to 30× more cycles) on ARM cores than on x86 cores. Contributing to this gap, we found that ARM cores executed 17.5× more instructions than x86 cores. We believe that most of the ARM-to-x86 gap for bwaves can be explained by large differences in the number of instructions required to complete the same amount of work. We performed detailed analysis to find the source of the instruction count discrepancies. To begin, we found from the execution profile that complex double floating point operations that the compiler translates to longer instruction sequences for ARM than for x86 are a significant source of additional instructions: 37% of all cycles for ARM cores are spent in __aeabi_dadd, and 29% of all cycles are spent in __aeabi_dmul, whereas neither of these routines appear in the x86 summary.

We use flags to force gcc to compile floating point instructions to SSE 2 (x86) and NEON (ARM) instructions. This decision is the fairest in general, since ARM's VFP unit is known to be significantly slower than the NEON unit for single precision floating point operations. However, unlike the VFP unit, the NEON unit is not IEEE754 compliant, and double precision operations are mapped to library calls. The result is that for ARM architectures, gcc—in the absence of FP relaxation—compiles double-precision floating point arithmetic to library calls that add significant overhead compared to short instruction sequences on x86. One solution to bwaves's outlier status would be to use different compiler flags for benchmarks with significant amounts of double precision arithmetic.

Key Finding 5: The microarchitecture has the dominant impact on performance. The ARM, x86, and MIPS architectures have similar instruction counts. The microarchitecture, not the ISA, is responsible for performance differences.

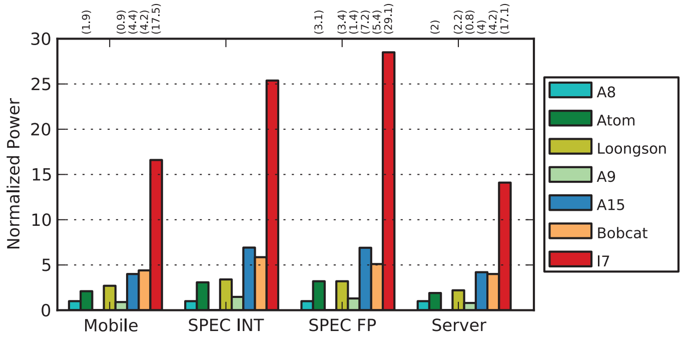


Fig. 8. Raw average power normalized to A8.

Step 6: *ISA Influence on Microarchitecture*

Key Finding 6: As shown in Table VIII, there are significant differences in microarchitectures. Drawing on instruction mix and instruction count analysis, we feel that the only case where the ISA forces larger structures is on the ROB size, physical rename file size, and scheduler size since there are almost the same number of x86 micro-ops in flight compared to ARM and MIPS instructions. The difference is small enough that we argue it is not necessary to quantify further. Beyond the translation to micro-ops, pipelined implementation of an x86 ISA introduces no additional overheads over an ARM or MIPS ISA for these performance levels.

5.2. Power and Energy Analysis

Step 1: *Average Power*

Data: Figure 8 shows average power normalized to the A8.[13] Per-benchmark data are shown in Figure 19 of Appendix I. Average ratios are in the included table.

Ratio	Mobile	SPEC INT	SPEC FP	Server
Atom/A8	2.1 (1.9)	3.1	3.2 (3.1)	1.9 (2)
i7/Loongson	6.2	7.5	9 (8.6)	6.5 (7.7)
i7/A9	17.8 (19.5)	17.3	22.8 (20.9)	18.6 (21.4)
i7/A15	4.1 (4)	3.7	4.1 (4)	3.4 (4)
i7/Bobcat	3.8 (4.2)	4.3	5.6 (5.4)	3.5 (4.1)

Finding E1: There are large differences in power across implementations, with i7 (x86) being the most power-hungry, followed by A15 (ARM) and Bobcat (x86).

Key Finding 7: Power consumption does not have a direct correlation to the ISA being RISC or CISC.

Step 2: *Average Technology Independent Power*

Data: Figure 9 shows technology-independent average power; cores are scaled to 1GHz at 45nm (normalized to A8). Per-benchmark data are shown in Figure 20 of Appendix I. Average ratios are in the included table.

[13] In this section, we normalize to A8 because it uses the least power.

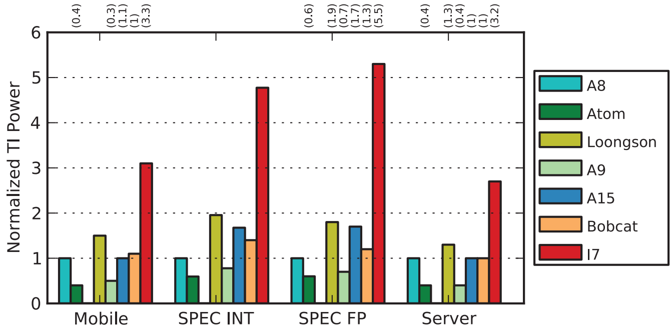


Fig. 9. Tech-independent average power normalized to A8.

Ratio	Mobile	SPEC INT	SPEC FP	Server
A8/Atom	2.5 (2.8)	1.7	1.7 (1.7)	2.7 (2.7)
A9/Atom	1.2 (1.3)	1.3	1.1 (1.2)	1.1 (1.1)
i7/Bobcat	3 (3.3)	3.4	4.4 (4.2)	2.8 (3.2)
A15/Bobcat	0.9 (1.1)	1.2	1.4 (1.3)	1.1 (1)
Loongson/Bobcat	1.5	1.4	1.5 (1.5)	1.3 (1.3)

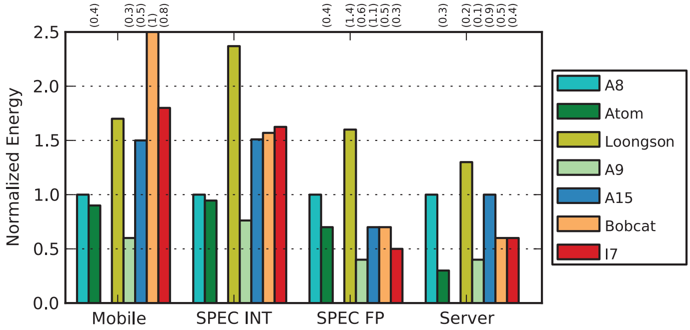


Fig. 10. Raw average energy normalized to A8.

Finding E2: With frequency and technology scaling, A8, A9, and Atom appear to be optimized for power and A15, Bobcat, and i7 optimized for performance.

Finding E3: Loongson appears to be neither a power optimized nor a performance optimized design. After scaling, Loongson stands out to be the second most power hungry while also being the third most cycle hungry implementation. This can be attributed to the more advanced microarchitectures of the ARM and x86 implementations, not the ISA.

Finding E4: ISA appears irrelevant among power and performance optimized cores: despite being from different ISAs, A9 and Atom are within 1.3× of each other. The same holds true for A15 and Bobcat.

Key Finding 8: The choice of power- or performance-optimized core designs impacts core power use more than does ISA.

Step 3: *Average Energy*

Data: Figure 10 shows energy (product of power and time). Per-benchmark data are shown in Figure 21 of Appendix I. Average ratios are in the included table.

Ratio	Mobile	SPEC INT	SPEC FP	Server
Atom/A8	0.9 (0.2)	0.9	0.7 (0.4)	0.3 (0.3)
i7/Loongson	1.1	0.7	0.3 (0.3)	0.5 (1.8)
i7/A9	3 (3)	2.1	1.2 (0.6)	1.5 (5.1)
i7/A15	1.2 (1.5)	1.1	0.7 (0.3)	0.6 (1.8)
i7/Bobcat	0.7 (0.8)	1	0.6 (0.7)	1 (0.9)

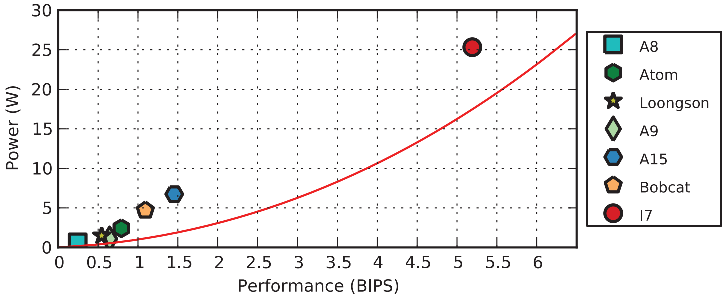


Fig. 11. Power performance tradeoffs.

Finding E5: Despite Atom and i7 being the most power hungry in-order and out-of-order implementations, Atom consumes less energy than A8, and i7 consumes less energy than Loongson, A15, and Bobcat, on most benchmarks. This shows that high-performance, high-power implementations (e.g., i7) can be energy efficient.

Finding E6: Across all implementations and benchmark suites, Loongson consumes the most energy.

Key Finding 9: Since power and performance are both primarily design choices, energy use is also primarily impacted by design choice. ISA's impact on energy is insignificant.

Step 4: *ISA Impact on Microarchitecture*

Data: Table VIII outlines microarchitecture features.

Finding E7: The energy impact of the ISA is that it requires micro-ops translation and an additional micro-ops cache. Furthermore, since the number of micro-ops is not significantly higher, the energy impact of x86 support is small.

Finding E8: Other power-hungry structures like a large L2-cache, highly associative TLB, aggressive prefetcher, and large branch predictor seem dictated primarily by the performance level and application domain targeted by the implementations and are not necessitated by ISA features.

5.3. Tradeoff Analysis

Step 1: *Power-Performance Tradeoffs*

Data: Figure 11 shows the geometric mean power-performance tradeoff for all benchmarks using technology node-scaled power. We generate a cubic curve for the power-performance tradeoff curve. Given our small sample set, a core's location on the frontier does not imply that it is optimal.

Data: For the table here, we choose A8 (RISC) and Atom (CISC) as representatives of their respective ISAs since they consume the least power among all implementations of the respective ISAs.

Ratio	Performance	Power
A15/A8	6	10.7
A9/A8	2.7	1.7
Loongson/A8	2.2	2.4
i7/Atom	6.6	10.5
Bobcat/Atom	1.4	2

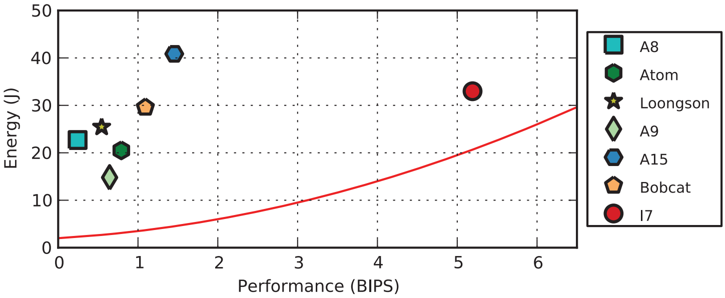


Fig. 12. Energy performance tradeoffs.

Key Finding 10: Regardless of ISA or energy-efficiency, high-performance processors require more power than lower-performance processors.[14] They follow well-established cubic power/performance tradeoffs regardless of ISA.

Step 2: *Energy-Performance Tradeoffs*

Data: Figure 12 shows the geometric mean energy-performance tradeoff using technology node- scaled energy. We generate a quadratic energy-performance tradeoff curve. Again, a core's location on the frontier does not imply optimality.

Finding T1: Balancing power and performance leads to energy-efficient cores, regardless of the ISA: A9 and Atom consume less energy and perform better than A8 and Loongson. The same holds true for i7 when compared to A15.

Data: We consider the Energy-Delay (ED) metric to capture both performance and power. Figure 13(a) shows the ED metric at various exponents. The exponent weighs the importance of performance over power.

Data: Figures 13(b) and 13(c) zoom into Figure 13(a) to depict the points where A15 and Bobcat cross over A9.

Finding T2: i7 has the best ED metric at all exponents.

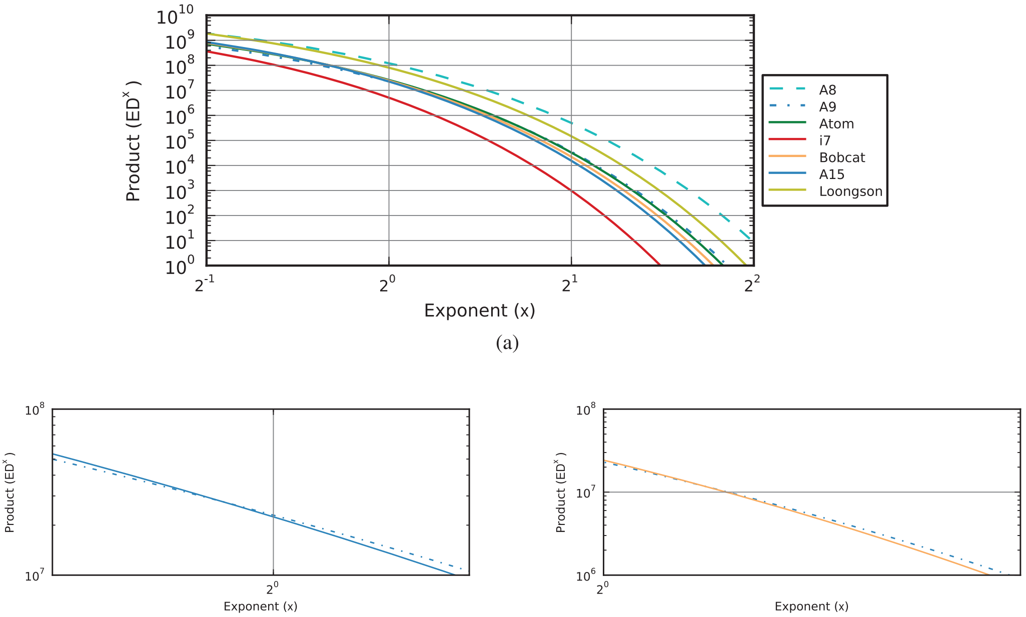
Finding T3: Up to $ED^{0.9}$, A9 (ARM) is a more balanced design than A15 (ARM) and Bobcat (x86). It loses that advantage after $ED^{0.9}$ and $ED^{1.2}$, irrespective of the fact that A15 and Bobcat represent different ISAs. Power-performance balance is determined by the microarchitecture, not the ISA.

Key Finding 11: It is the microarchitecture and design methodologies that really matter.

6. CHALLENGES

During this study, we encountered infrastructure and system challenges, missteps, and software/hardware bugs. Table XIII outlines these issues as a potentially useful guide for similar studies. We describe them in more detail here.

[14] Loongson consumes more power than A9 while performing worse. This implies design inefficiencies in Loongson's microarchitecture.



(a)

(b) A15-A9 Crossover

(c) Bobcat-A9 Crossover

Fig. 13. Energy delay tradeoffs.

Table XIII. Summary of Challenges

Challenge	Description
Board Cooling (A8,A9,A15)	No active cooling leading to failures **Fix:** Use a fan-based laptop cooling pad
Networking (A8,A9)	ssh connection used up to 20% of CPU **Fix:** Use a serial terminal
Networking (Atom)	USB networking not supported **Fix:** Use as standalone terminal
Perf Counters (A8,A9)	PMU poorly supported on selected boards **Fix:** Backport over 150 TI patches to A8 and A9. Linux 3.8 used on A15.
Perf Counters (A15)	Linux 2.6 poorly supported for A15. **Fix:** Used Linux 3.8 instead.
Perf Counters (Loongson)	Not enough interrupt ports available to support all required perf counters. **Fix:** Enabled only two perf counters.
Compilation (A8,A9)	Failures due to dependences on >100 packages **Fix 1:** Pick portable equivalent (lighttpd) **Fix 2:** Work through errors (CLucene & WebKit)
Tracing (A8,A9)	No dynamic binary emulation **Fix:** Use gem5 to generate instruction traces

Board cooling. The ARM boards lack active cooling and repeatedly rebooted due to overheating while under test. A fan-based laptop cooling pad fixed the problem.

Network over USB. The ssh connection to the A8 and A9 boards used up to 20% of the CPU, which was unsuitable for performance benchmarking. We instead used a serial terminal to access these boards. The Atom board does not support USB networking; hence, we used it as a standalone terminal. A15 and Bobcat boards were also used as standalone terminals.

Microprocessor PMU infrastructure. The performance counters on the ARM processor are poorly supported on community-supported boards. We backported over 150 TI patches to the Linux kernel 2.6 to support performance counters and PMU interrupts on A8 and A9. On A15, we decided to use Linux 3.8 due to technical difficulties porting the 2.6 kernel. On Loongson, due to limited interrupt ports on the processor, only two performance counters could be enabled. We chose to use them to count cycles and instructions.

Compilation. For simple benchmarks like SPEC that rely on `libc`, `gcc` works remarkably well as a cross-platform compiler. However, for the ARM environment, the compiler often fails when compiling complex code bases that have not been rigorously tested on Linux due to dependences on more than 100 packages. Overcoming these linking errors is a tremendously tedious process. We either carefully choose equivalent highly portable workloads (e.g., `lighttpd`) or worked through the errors (e.g., CLucene and WebKit).

Tracing and debugging. ARM open-source tracing infrastructure is limited and lacks dynamic binary translation tools like Pin or DynamoRIO. `ptrace`-based approaches were too slow; QEMU correctly emulated, but its JIT obfuscated the instruction stream. We used gem5 for ARM traces; gem5 does not support all benchmarks (e.g., `lighttpd`). For MIPS, binary instrumentation tools do not exist.

7. CONCLUSION

In this work, we revisit the RISC versus CISC debate considering contemporary MIPS, ARM, and x86 processors running modern workloads to understand the role of ISA on performance, power, and energy. Our study suggests that the ISA being RISC or CISC is irrelevant, as summarized in Table XIV, which includes a key representative quantitative measure for each analysis step. For a generation of students, junior researchers, and others not exposed to the 90s RISC/CISC debate, our article is a rigorous and thorough analysis that confirms in an experimental fashion that ISA being RISC or CISC is irrelevant. For the experts, the detailed quantification and measurements add significant value: We have made all data from the measurements available for others to analyze and interpret based on other metrics.

Our work includes a methodological contribution. During this study, we encountered infrastructure and system challenges, missteps, and software/hardware bugs. Table XIII outlines these issues as a potentially useful guide for similar studies.

We conclude by reflecting on whether there are certain metrics for which RISC or CISC matters, and we place our findings in the context of past ISA evolution and future ISA and microarchitecture evolution.

Area overheads and performance. Considering area normalized to the 45nm technology node, we observe that A8's area is 4.3mm^2, AMD's Bobcat's area is 5.8mm^2, A9's area is 8.5mm^2, and Intel's Atom is 9.7mm^2.[15,16,17] The smallest, A8, is smaller than Bobcat by 25%. We feel much of this is explained by simpler core design (in-order vs. out-of-order), smaller caches, predictors, and TLBs. We also observe that A9's area is in-between Bobcat and Atom and is close to Atom's. Further detailed analysis is

[15] http://www.cs.virginia.edu/~skadron/cs8535_s11/ARM_Cortex.pdf.

[16] chip-architect, http://www.chip-architect.com/news/AMD_Ontario_Bobcat_vs_Intel_Pineview_Atom.jpg.

[17] Improved ARM core, other changes in TI mobile app processor, http://www.cs.virginia.edu/~skadron/cs8535_s11/ARM_Cortex.pdf.

Table XIV. Summary of Findings

				Representative
	#	Finding	Support	Data: A8/Atom
Performance	1	Large performance gaps exist	Figure 2	2× to 152×
	2	Cycle-count gaps are less than 3× (A8 to Atom, OoO Processors to i7)	Figure 3	≤2×
	3	CPI can be less on x86 implementation	Figures 3 and 4	A8: 2.5, Atom: 1.9
	4	ISA performance effects indistinguishable between x86 and ARM	Table VII Figures 5 and 6	inst. mix same short x86 insts
	5	μ architecture, not the ISA, responsible for performance differences	Tables IX, X, XI, XII	324× Br MPKI
	6	Beyond micro-op translation, x86 ISA introduces no overheads over ARM and MIPS ISA	Table VIII	
Power	7	Power consumption does not have a direct correlation to ISA	Figure 8	Atom/A8 raw power: 3×
	8	Choice of power or perf. optimization impacts power use more than does ISA	Figure 9	Atom/A8 power @1 GHz: 0.6×
	9	Energy use primarily a design choice; ISA's impact insignificant	Figure 10	Atom/A8 raw energy: 0.8×
Trade-offs	10	High-performance processors require more power than lower-performance processors	Figure 11	A9/A8: 1.7× i7/Atom: 10.5×
	11	It is the μ-architecture and design methodology that really matters	Figures 12 and 13	i7 best for all ED exponents

required to determine the individual contributions of the ISA and the microarchitecture structures to these differences.

A related issue is the performance level for which our results hold. Considering very-low-performance processors like the RISC ATmega324PA microcontroller with operating frequencies from 1 to 20 MHz and power consumption between 2 and 50mW,[18] the overheads of a CISC ISA (specifically the complete x86 ISA) are clearly untenable. In similar domains, even ARM's full ISA is too rich; the Cortex-M0, meant for low-power embedded markets, includes only a 56-instruction subset of Thumb-2. Our study suggests that at performance levels in the range of A8 and higher, RISC/CISC is irrelevant for performance, power, and energy. Determining the lowest performance level at which the RISC/CISC ISA effects are irrelevant, for all metrics, is interesting future work.

Role of ISA. Although our study shows that RISC and CISC ISA traits are irrelevant to power and performance characteristics of modern cores, ISAs continue to evolve to better support exposing workload-specific semantic information to the execution substrate. On x86, such changes include the transition to Intel64 (larger word sizes, optimized calling conventions, and shared code support), wider vector extensions like AVX, integer crypto and security extensions (NX), hardware virtualization extensions, and, more recently, architectural support for transactions in the form of HLE. Similarly, ARM ISA has introduced shorter fixed-length instructions for low-power targets (Thumb), vector extensions (NEON), DSP and bytecode execution extensions (Jazelle DBX), Trustzone security, and hardware virtualization support. Thus, although ISA

[18] Atmel Datasheet, http://www.atmel.com/Images/doc2503.pdf.

evolution has been continuous, it has focused on enabling specialization and has been largely agnostic of RISC or CISC. Other examples from recent research include extensions to allow the hardware to balance accuracy and reliability with energy efficiency [Esmaeilzadeh et al. 2012; de Kruijf et al. 2010] and extensions to use specialized hardware for energy efficiency [Govindaraju et al. 2011]. Finally, Venkat and Tullsen [2014] recently studied the performance and energy benefits of ISA heterogenity. Our interpretation of their data, experiments, and analysis is the following: The presence or absence of specializations such as floating point and SIMD support, on one ISA over the other, are the primary ISA differentiators for performance and energy. In essence, we argue RISC versus CISC was about the *syntax of expression* to the machine—arguably important before, but irrelevant now. In the future, the role of the ISA is going to be dominated by *expressing richer semantics*. This is the important and broad implication of our work for related fields of systems, compiler, and application development.

It appears that decades of hardware and compiler research has enabled efficient handling of both RISC and CISC ISAs, and both are equally positioned for the coming years of energy-constrained innovation.

A. APPENDIX I: DETAILED COUNTS

Data: Figure 14(a) shows dynamic instruction (macro) counts for Loongson (MIPS), A8 (ARM), and Atom (x86) normalized to A8 macro-instructions.

Data: Figure 14(b) shows dynamic micro-op counts for Bobcat, Atom, and i7 normalized to A8 macro-instructions.

Data: Table XV(a) shows average MIPS, ARM, and x86 static binary sizes, measuring only the binary's instruction segment.

Data: Table XV(b) shows average dynamic MIPS, ARM, and x86 instruction lengths.

Data: Table XVI shows per-benchmark CPI.

Data: Figure 15 shows per-benchmark execution time normalized to i7.

Data: Figure 16 shows per-benchmark branch mispredictions per 1,000 ARM instructions for ARM and x86 platforms.

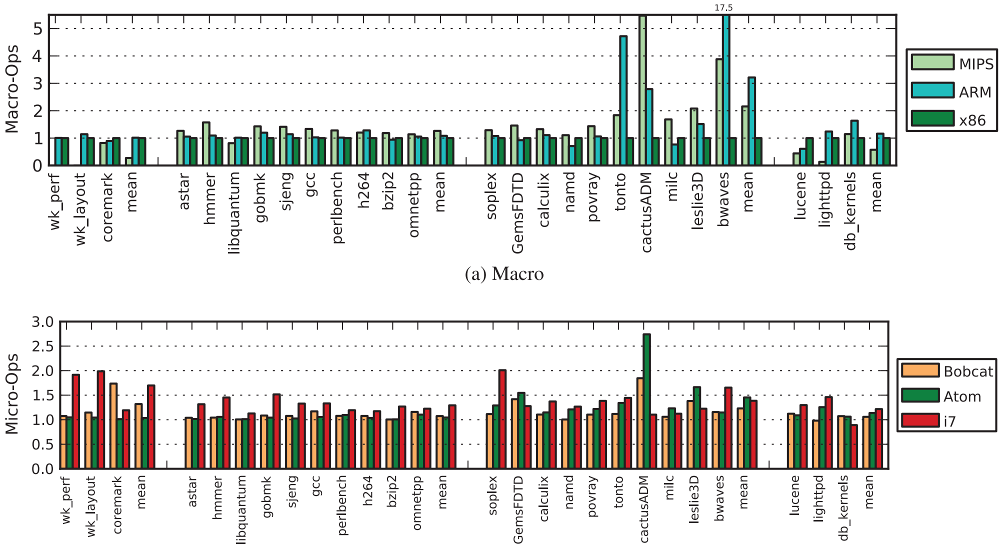
Data: Figure 17 shows per-benchmark L1 data cache misses per 1,000 ARM instructions for ARM and x86 platforms.

Data: Figure 18 shows per-benchmark instruction cache misses per 1,000 ARM instructions for ARM and x86 platforms.

Data: Figure 19 shows average power normalized to A8.

Data: Figure 20 shows technology-independent average power; cores are scaled to 1GHz at 45nm (normalized to A8).

Data: Figure 21 shows average energy normalized to A8.



(a) Macro

(b) Micro

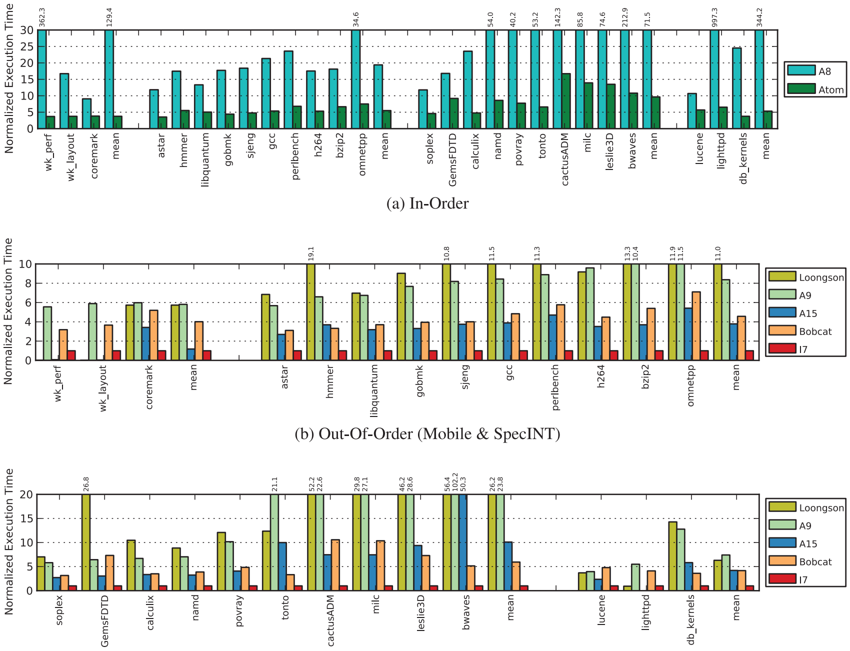
Fig. 14. Instructions normalized to Atom macro-ops.

Table XV. Instruction Size Details: (a) Static Binary (MB), (b) Average Dynamic Instruction (B)

		Mobile			SPEC INT										SPEC FP										Server		
		coremark	wk - layout	wk - perf	astar	libquantum	hmmer	h264	gobmk	bzip2	sjeng	gcc	perlbench	omnetpp	soplex	GemsFDTD	calculix	povray	tonto	namd	leslie3D	milc	cactusADM	bwaves	lucene	db_kernel	lighttpd
Bin	MIPS	0.5	—	—	0.7	0.7	1	1.7	2.2	0.6	0.7	4.4	2	1.9	1.4	1.2	2.9	2.2	5.2	1	0.9	0.8	1.7	0.8	0.8	0.5	0.2
	ARM	0.02	1.3	1.3	0.6	0.6	0.9	1.7	2.1	0.5	0.6	3.9	1.9	2.0	1.4	1.3	2.7	2.0	4.8	0.9	0.9	0.7	1.7	0.8	0.4	0.47	0.1
	x86 - gcc	0.02	1.4	1.4	0.7	0.7	0.9	1.5	2.1	0.7	0.7	4.1	1.7	1.7	1.5	1.3	2.6	1.8	5.2	0.9	0.9	0.7	1.5	0.8	1.0	0.6	0.2
	x86 - icc	0.6	1.5	1.5	0.7	0.7	1.0	1.3	2.2	0.7	0.8	4.3	1.9	2.2	1.5	1.7	3.1	2.2	6.8	1.0	1.4	0.8	2.0	—	—	—	—
Inst	MIPS	4.0	—	—	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0
	ARM	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0
	x86 - gcc	2.4	3.7	3.7	2.9	3.0	3.0	3.5	3.1	3.6	3.5	2.8	2.9	2.7	2.7	3.4	2.9	2.6	3.4	3.3	4.1	2.6	6.4	3.0	3.7	2.6	3.7
	x86 - icc	2.5	3.2	3.2	3.2	2.9	3.6	3.3	3.3	3.4	3.6	2.9	3.2	2.8	3.1	3.6	3.3	3.5	4.2	4.9	5.0	4.1	6.1	—	—	—	—

Table XVI. Cycles per Instruction (CPI) per Benchmark

	Mobile			SPEC INT										SPEC FP										Server		
	coremark	wk - layout	wk - perf	astar	libquantum	hmmer	h264	gobmk	bzip2	sjeng	gcc	perlbench	omnetpp	soplex	GemsFDTD	calculix	povray	tonto	namd	leslie3D	milc	cactusADM	bwaves	lucene	db_kernel	lighttpd
MIPS	1.1	—	—	1.5	0.8	1.6	1	1.3	1.7	1.3	1.5	1.5	1.4	1.3	3.2	1.3	1.2	1.3	1.1	3.3	2.1	1.4	2.2	1.4	1.5	2.2
ARM	1.2	3.4	77	2.3	0.9	1.5	1.3	2.3	2.2	2	2.7	2.9	3.4	1.9	2.4	2.6	4.1	1.7	7.5	5.4	10.2	5.7	1.4	2.2	1.4	190
x86	1.2	2.4	2.2	2	1	1.5	1.4	1.9	2.1	1.6	1.9	2.3	2.2	2.3	3.4	1.6	2.3	2.7	2.3	4.1	3.5	5.2	3.4	2	0.9	4.3

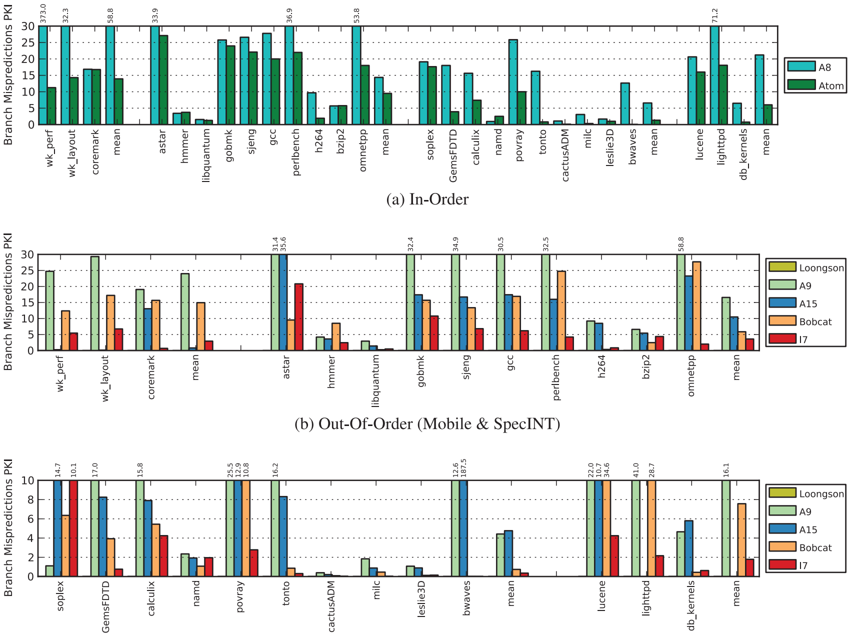


(a) In-Order

(b) Out-Of-Order (Mobile & SpecINT)

(c) Out-Of-Order (SpecFP & Server)

Fig. 15. Per-benchmark execution time normalized to i7.

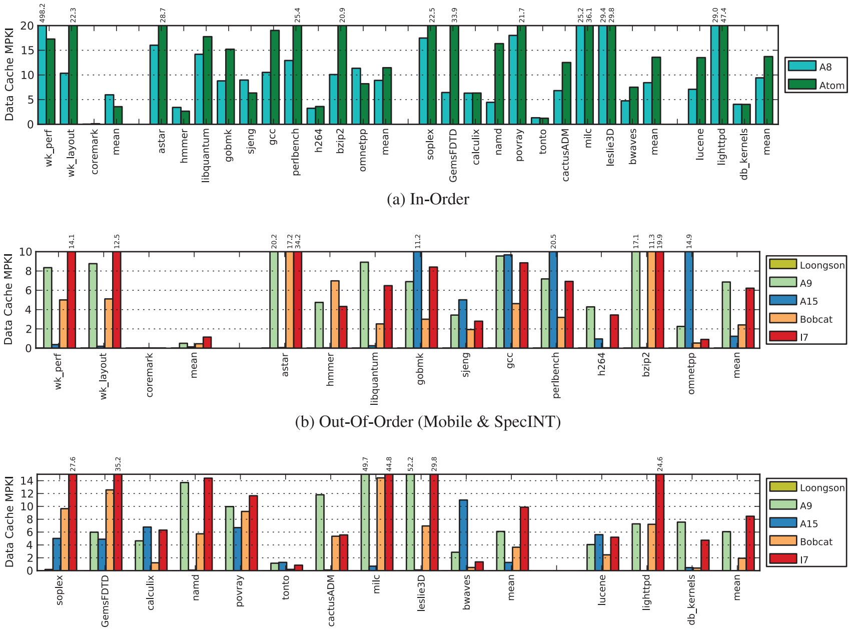


(a) In-Order

(b) Out-Of-Order (Mobile & SpecINT)

(c) Out-Of-Order (SpecFP & Server)

Fig. 16. Per-benchmark branch mispredictions per 1,000 ARM instructions.

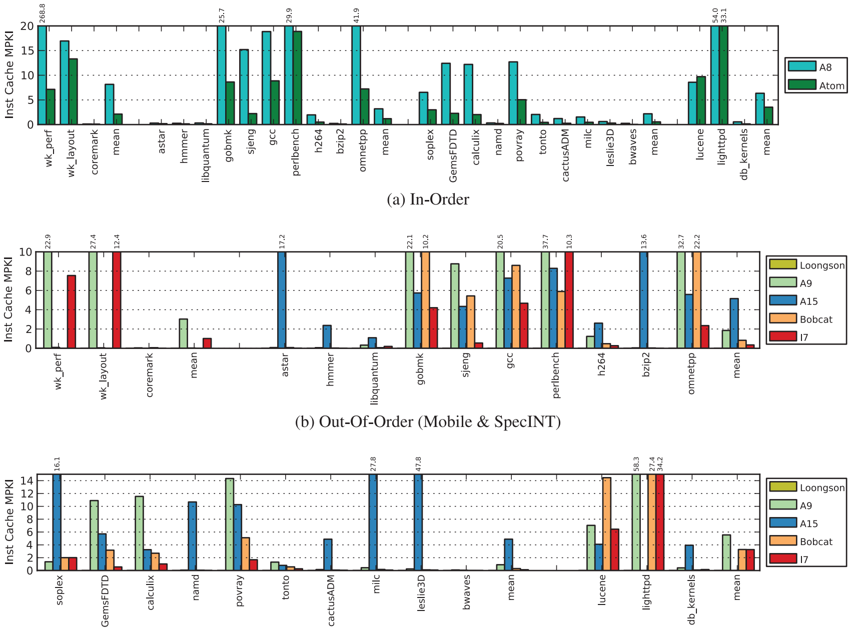


(a) In-Order

(b) Out-Of-Order (Mobile & SpecINT)

(c) Out-Of-Order (SpecFP & Server)

Fig. 17. Per-benchmark L1 data cache misses per 1,000 ARM instructions.

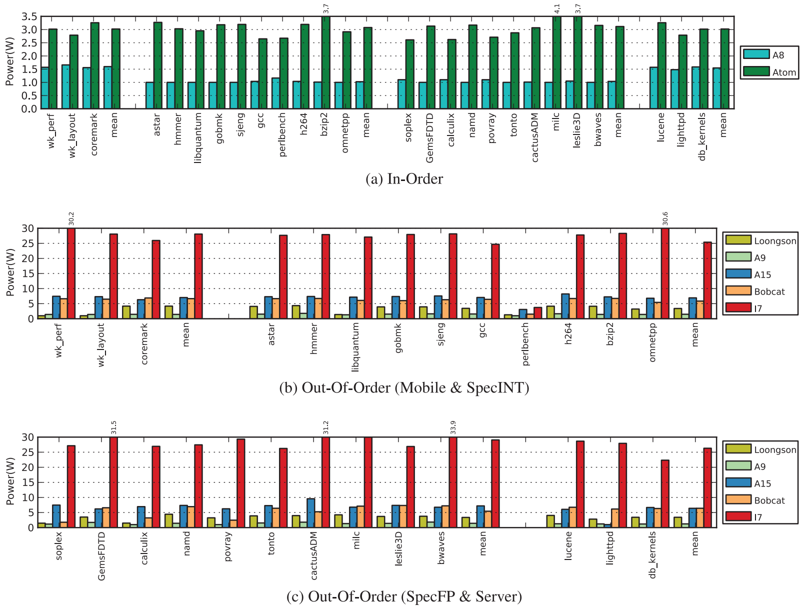


(a) In-Order

(b) Out-Of-Order (Mobile & SpecINT)

(c) Out-Of-Order (SpecFP & Server)

Fig. 18. Per-benchmark instruction cache misses per 1,000 ARM instructions.

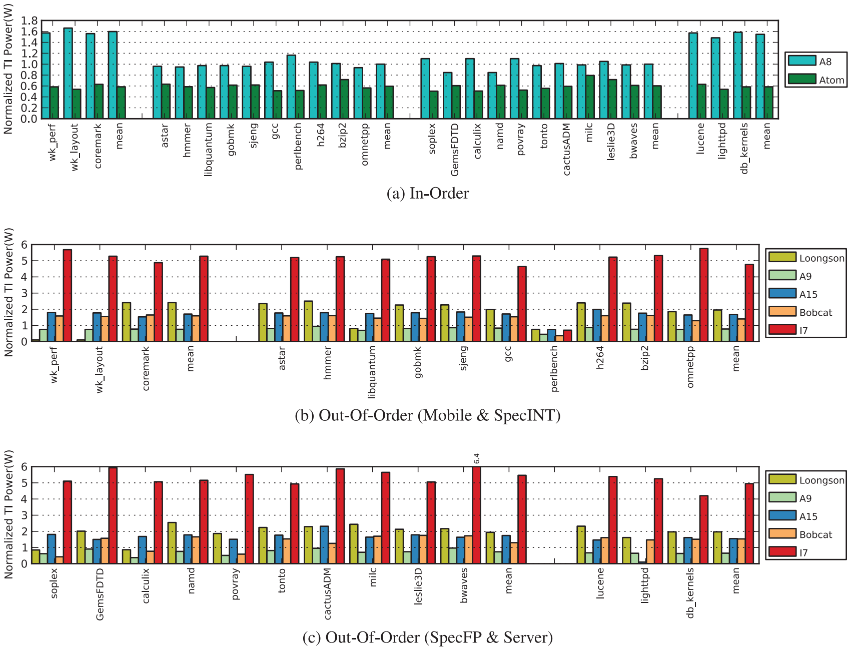


(a) In-Order

(b) Out-Of-Order (Mobile & SpecINT)

(c) Out-Of-Order (SpecFP & Server)

Fig. 19. Per-benchmark raw average power normalized to A8.

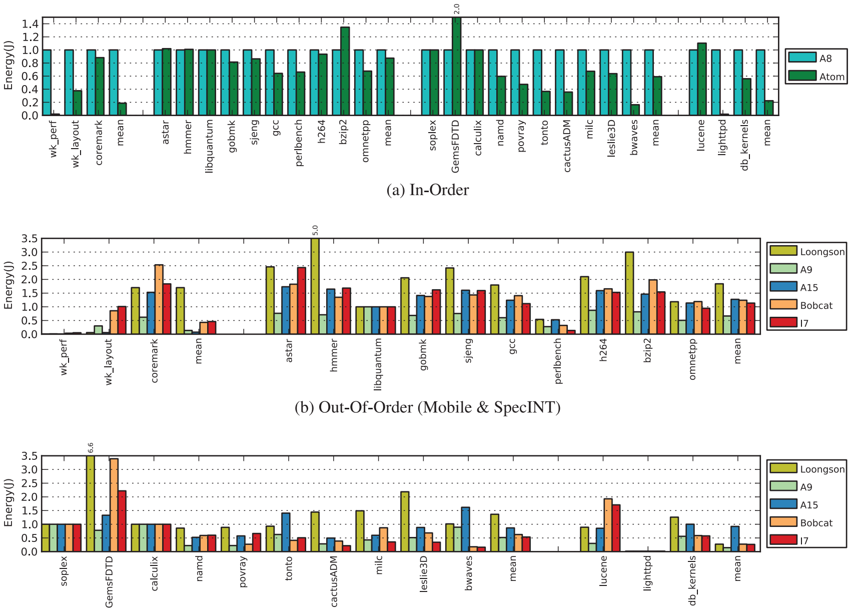


(a) In-Order

(b) Out-Of-Order (Mobile & SpecINT)

(c) Out-Of-Order (SpecFP & Server)

Fig. 20. Per-benchmark tech-independent average power normalized to A8.



(a) In-Order

(b) Out-Of-Order (Mobile & SpecINT)

(c) Out-Of-Order (SpecFP & Server)

Fig. 21. Per-benchmark raw average energy normalized to A8.

ACKNOWLEDGMENTS

We thank the anonymous reviewers, the Vertical group, and the PARSA group for comments. Thanks to Doug Burger, Mark Hill, Guri Sohi, David Wood, Mike Swift, Greg Wright, Jichuan Chang, and Brad Beckmann for comments on the article and thought-provoking discussions on ISA impact. Thanks for various comments on the article and valuable input on ISA evolution and area/cost overheads of implementing CISC ISAs provided by David Patterson.

REFERENCES

Max Baron. 2010. The single-chip cloud computer. *Microprocessor Report* (April 2010).

Dileep Bhandarkar. 1997. RISC versus CISC: A tale of two chips. *SIGARCH Computer Architecture News* 25, 1 (March 1997), 1–12.

Dileep Bhandarkar and Douglas W. Clark. 1991. Performance from architecture: Comparing a RISC and a CISC with similar hardware organization. In *ASPLOS'91*. 310–319.

Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark Hill, and David Wood. 2011. The gem5 simulator. *SIGARCH Computer Architecture News* 39, 2 (Aug. 2011), 1–7.

W. Lloyd Bircher and Lizy K. John. 2008. Analysis of dynamic power management on multi-core processors. In *ICS'08*. 327–338.

Emily Blem, Jaikrishnan Menon, and Karthikeyan Sankaralingam. 2013. Power struggles: Revisiting the RISC vs. CISC debate on contemporary ARM and x86 architectures. In *HPCA'13*. 1–12.

Derek Bruening, Timothy Garnett, and Saman Amarasinghe. 2003. An infrastructure for adaptive dynamic optimization. In *CGO'03*. 265–275.

Robert Colwell, Charles Y. Hitchcock, III, E. Jensen, H. Brinkley Sprunt, and Charles Kollar. 1985. Instruction sets and beyond: Computers, complexity, and controversy. *Computer* 18, 9 (Sept. 1985), 8–19.

Marc de Kruijf, Shuou Nomura, and Karthikeyan Sankaralingam. 2010. Relax: An architectural framework for software recovery of hardware faults. In *ISCA'10*. 497–508.

Hadi Esmaeilzadeh, Ting Cao, Yang Xi, Stephen Blackburn, and Kathryn McKinley. 2011. Looking back on the language and hardware revolutions: Measured power, performance, and scaling. In *ASPLOS'11*. 319–332.

Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. 2012. Architecture support for disciplined approximate programming. In *ASPLOS'12*. 301–312.

Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. 2012. Clearing the clouds: A study of emerging scale-out workloads on modern hardware. In *ASPLOS'12*. 37–48.

Michael J. Flynn, Chad L. Mitchell, and Johannes M. Mulder. 1987. And now a case for more complex instruction sets. *Computer* 20, 9 (1987), 71–83.

Venkatraman Govindaraju, Chen-Han Ho, and Karthikeyan Sankaralingam. 2011. Dynamically specialized datapaths for energy efficient computing. In *HPCA'11*. 503–514.

Anthony Gutierrez, Ronald G. Dreslinski, Thomas F. Wenisch, Trevor Mudge, Ali Saidi, Chris Emmons, and Nigel Paver. 2011. Full-system analysis and characterization of interactive smartphone applications. In *IISWC'11*. 81–90.

K. Hoste and L. Eeckhout. 2007. Microarchitecture-independent workload characterization. *IEEE Micro* 27, 3 (2007), 63–72. DOI:http://dx.doi.org/10.1109/MM.2007.56

Canturk Isci and Margaret Martonosi. 2003. Runtime power monitoring in high-end processors: Methodology and empirical data. In *MICRO'03*. 93.

Ciji Isen, Lizy John, and Eugene John. 2009. A tale of two processors: Revisiting the RISC-CISC debate. In *2009 SPEC Benchmark Workshop*. 57–76.

Changkyu Kim, Tim Kaldewey, Victor W. Lee, Eric Sedlar, Anthony D. Nguyen, Nadathur Satish, Jatin Chhugani, Andrea Di Blas, and Pradeep Dubey. 2009. Sort vs. hash revisited: Fast join implementation on modern multi-core CPUs. *VLDB'09* (2009), 1378–1389.

David A. Patterson and David R. Ditzel. 1980. The case for the reduced instruction set computer. *SIGARCH Comp. Arch. News* 8, 6 (1980), 25–33.

Jun Rao and Kenneth A. Ross. 2000. Making B+− trees cache conscious in main memory. In *SIGMOD'00*. 475–486.

Ashish Venkat and Dean M. Tullsen. 2014. Harnessing ISA diversity: Design of a heterogeneous-ISA chip multiprocessor. In *ISCA'14*. 121–132.

Received December 2013; revised June 2014; accepted October 2014