Building First-Order Energy Modeling Intuition in Computer Architecture Lectures

Christopher Torng
University of Southern California, Los Angeles, CA

Abstract

Computer architecture students today arguably do not have as close a connection to energy as they do to performance. Specifically, they are not trained to reason about energy in a *quantifiable* way. Architecture students are taught that performance and energy are considered equally important. However, most course material focuses on a performance-driven narrative, meaning that students learn about processors, memories, networks, and systems with a strong sense of the performance implications, but with little intuition for the energy implications. On the other hand, VLSI students concretely learn about energy, but they are immersed in a world of gates and transistors and do not connect their understanding about energy back up to the abstraction of SoC-level components. How can we bridge this gap to enable architecture students to reason about the energy implications of hardware design concepts, directly within a computer architecture class? In this work, we attempt to mitigate these challenges by introducing a teaching methodology that integrates energy into pipeline diagrams. We include examples of specific classroom tools including the representation of an energy map, and we show how pipeline diagrams augmented with an energy map can enable first-order quantitative comparisons of performance, energy, and power across different design points. The approach is simple enough for lecture, in-class activities, and in exams. We hope this approach can train future students in thinking from first principles in evaluating performance-energy tradeoffs.

ACM Reference Format:
Christopher Torng, University of Southern California, Los Angeles, CA . 2023. Building First-Order Energy Modeling Intuition in Computer Architecture Lectures . In *Workshop on Computer Architecture Education (WCAE '23), June 17, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3605507.3610632

1 Introduction

Energy is as important as performance in computer architecture, and this has been widely accepted for decades in both academia and industry. However, architecture students trained around the world in our university classes arguably do not have as close a connection to energy as they do to performance. We attribute this to a gap in traditional progression within computer engineering courses. Students in a computer architecture course are trained during lectures in microarchitectural concepts (e.g., pipelining, out-of-order execution, branch prediction, multithreading), but they are primarily provided tools to reason about performance (e.g., pipeline diagrams) and there is no similar first-order tool for energy. This means that most architecture students have little guiding intuition when asked how much energy it costs to make design decisions (e.g., add a branch predictor to a processor). These students do eventually expand their understanding about energy, but often primarily *after leaving architecture class*, typically in a digital VLSI class. However, such courses immerse students in a very different abstraction with gates, transistors, and wires. While the energy of a capacitor may now be very concrete, there are few curriculums where the same level of energy intuition is connected back to the abstraction of SoC-level components previously described in architecture class.

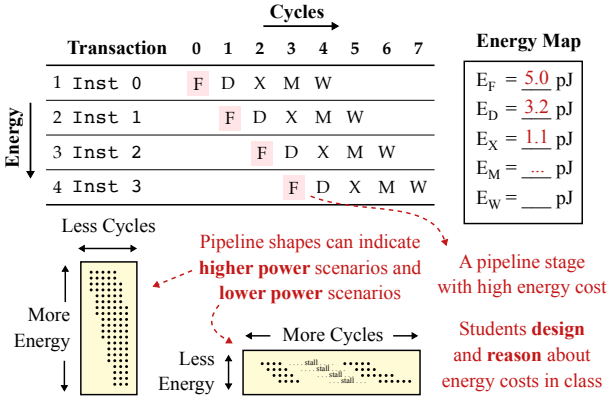


Figure 1: Integrating Energy into Pipeline Diagrams — Students build intuition about energy by associating more transactions with increasing energy consumption. Together with an energy map which breaks the energy down further, students can visualize high-level tradeoffs in performance, energy, and power directly within a lecture.

The key insight made in this paper is that energy can be added into pipeline diagrams in order to enable performance-energy tradeoff evaluations directly in computer architecture lectures. Simply put, we will enable energy to be discussed quantitatively at a whiteboard during lecture. One way to view the basic intuition is that pipeline diagrams are often drawn with one axis advancing forwards in time in units of cycles, while the other axis represents transactions. Transactions are often instructions in the context of a processor, but they may also be packets in the context of a network. More transactions represents more work, and therefore the axis visually depicts increasing energy consumption. While we are certainly not the first to observe the potential of this insight for teaching purposes, the idea has not been built upon and integrated in most curriculums to tap into its potential for education.

WCAE '23, June 17, 2023, Orlando, FL, USA

ACM ISBN 979-8-4007-0253-2/23/06.
https://doi.org/10.1145/3605507.3610632

Paper Artifact: https://github.com/ctorng/torng-energy-intuition-wcae2023

We build from the concept of integrating energy into pipeline diagrams and take steps to more concretely visualize energy. Figure 1 illustrates our approach of associating each unique letter within the pipeline diagram with a per-access energy cost. This energy is counted each time the letter appears, representing the hardware resources within that stage. There is a procedure to assign these numbers, and we prepare students to reason about relative energy costs in class. The central teaching construct that results is the *energy map*, a simple component-based energy model that is shown already built in Figure 1. Once completed, the energy map enables students to "see" energy directly from letters in their pipeline diagrams (e.g., which letters are more expensive?). This allows architecture students to reason about the energy implications of different hardware design decisions from their pipeline diagrams, directly within a computer architecture lecture.

Designing a course that builds upon this initial intuition can be done in many ways. In this paper, we illustrate one methodology that was used in a classroom setting in computer architecture courses taught at USC. We include examples of specific classroom tools including the representation of an energy map, pipeline diagrams for in-order and out-of-order cores, illustrations of energy and performance across these design points which can now be compared quantitatively in lecture, in-class activities with power-performance plots, an exam question, and also a methodology to grade such questions so as to properly drive the feedback loop in a student's mind that convinces them of their own mastery in critical thinking. We hope that our examples can help develop stronger energy narratives in future computer architecture courses by integrating and building intuition about energy into lecture notes.

2 Background and Goals

Energy modeling is a core element of supporting infrastructure for computer systems design-space exploration in both academia and industry. This section overviews the merits of existing infrastructure for energy modeling from a lecture-oriented perspective.

2.1 Research-Grade Energy Modeling Tools

The computer architecture community has developed various energy modeling research tools and approaches to model energy at various different levels of abstraction. Energy has been modeled at the component level (e.g., CACTI [12] for SRAMs and memory macros), at the processor level (e.g., McPAT [11], Wattch [4]), and at the design-specific level typically for hardware accelerator designs using component-based energy lookup tables [2, 15]. Researchers have also built early design-space exploration frameworks for domain-specific accelerators which also adopt table-based approaches including Aladdin [13] and Accelergy [16]. It is worth noting that because of the difficulty in creating accurate high-level energy models, recent works are increasingly relying directly on detailed gate-level power estimation, especially for domain-specific hardware accelerators [5, 10, 14].

Are energy modeling tools used in computer architecture research suitable for a lecture environment? — While these tools are powerful, they are designed to estimate energy at scale across tens, hundreds, or even thousands of workload traces. Instead of building intuition for how to reason about energy, they are instead designed to produce large datasets of results, which can be analyzed to gain insights about a given design space. However, just as pipeline diagrams are taught in lectures and then extrapolated to large-scale performance models using tools like gem5 [3] in research labs, the community needs a similar first-order energy modeling approach suitable for a whiteboard before students begin to use large-scale energy modeling tools.

2.2 Course Textbooks

Some literature is explicitly written for the classroom in the form of course textbooks which discuss power and energy. Hennessy and Patterson [7] is the classic textbook in the field, and others exist as well including Dubois [6] which can be used in an advanced course. These textbooks often introduce power and energy coupled with an overview of semiconductor technology trends. The texts may even dedicate an entire chapter to the fundamental first-order energy and power equations, including $E = \frac{1}{2}CV^2$ and $P_D = \frac{1}{2}\alpha CV^2F$, and their implications on architecture [6]. However, once computer architecture concepts are explored in detail, the discussion is driven primarily by performance metrics, and energy is only discussed in passing. One indication of this is that there are very few power (or energy) versus performance plots in either Hennessy and Patterson or Dubois in the discussion of pipelining, caching, out-of-order execution, vector, VLIW, multithreading, and other classic techniques. We attribute this to the lack of tooling available for students to gain intuition about energy, in contrast to performance. Finally, some research literature such as Horowitz [8] include quantified absolute energy numbers (e.g., instruction energy breakdowns) in a 45 nm technology, with Jouppi et al. updating these numbers in a 7 nm technology [9]. While very useful, it is not immediately obvious how to connect these numbers into the classroom.

How can we integrate more energy-driven discussion into lecture without completely re-writing our course textbooks? — We will likely continue to rely on these well-written classic textbooks in our teaching. However, in order to integrate more energy-driven discussion into class, we may be forced to develop new material with little support from the texts. We make a key observation that enables both needs to be met. We note that pipeline diagrams are already prevalent in most existing course textbooks. By *specifically* building a first-order energy model on top of these diagrams (as opposed to some other device), we can freely augment energy-driven discussion wherever a pipeline diagram is already present. This allows us to maintain a parallel narrative about energy, while still following along with the existing classic textbooks.

2.3 Design Experiences in the Classroom

Many university courses include design experiences that are explicitly designed for the classroom. These experiences simplify the infrastructure to run research-grade performance and energy simulators such as gem5 [3]. Students may even work with FPGAs and ASIC tool flows. These tools can generate performance and energy reports for each hardware design point in the project.

Are teaching-grade course projects built around simulators already sufficient to train students in building intuition about energy? — Design experiences have various weaknesses when deployed on their own, without a corresponding first-order

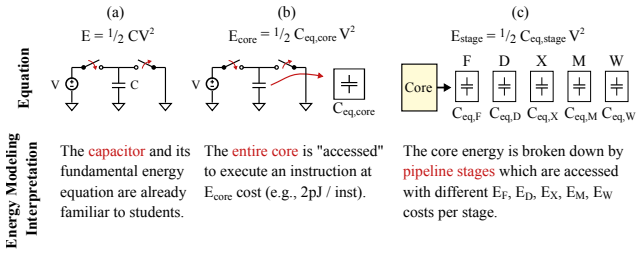


Figure 2: Energy Intuition Road Map — We build intuition for energy from (a) the fundamental energy equation for an ideal capacitor, (b) the insight of viewing a processor core as some $C_{eq,core}$, and (c) breaking the hardware down further into pipeline diagram letters with different $C_{eq,stage}$ and different energies per access.

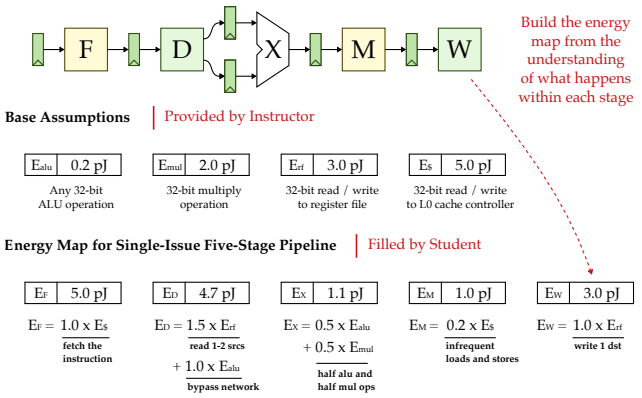


Figure 3: Example Energy Map for Five-Stage Pipelined Processor — Energy maps allow students to reason about the energy costs of pipeline diagram letters. This example shows energy costs for E_F, E_D, E_X, E_M, and E_W using the provided base assumptions. Students will justify each term.

model introduced in the lectures. The most glaring sign of students lacking intuition is the tendency to believe the tool and to write down exactly what the tool said (e.g., "this design point is better than that design point, because the tool reports their powers as X milliwatts and Y milliwatts and X is smaller than Y"). Unfortunately, tools can be used incorrectly, and the more problematic part is that these design experiences often emphasize more how to use the specific tools, as opposed to teaching students how to reason about energy. Another weakness is that design experiences are usually designed around a specific, narrow design space (e.g., a simple RISC-V core) due to the challenges in developing high-quality teaching infrastructure. Many architecture concepts simply cannot be covered and are left to the side. Finally, in most universities, course lectures occupy more time than labs do in terms of the total amount of time in which students engage with the course material. In order for energy to be considered at equal status with performance, the lectures themselves must be designed to make the priorities clear.

2.4 Industry-Grade Energy Modeling Tools

Lectures that teach a first-order energy modeling approach will ideally transition cleanly to commercial tools that students are likely to use after joining the workforce. These detailed, industry-grade energy models for complex ASIC designs are composed of gate-level netlists, per-net activities, technology constraints and libraries, and commercial power estimation tools [1]. To a large extent, these commercial tools are built upon the same component-based energy modeling approach as we describe in this work. Our abstractions are simply at a higher level. For example, students count pipeline diagram letters (instead of per-net activities) and use energy maps (similar to liberty files) for per-access energy costs. The approach is largely hierarchical and lends itself to clean transitions to industry-standard methodologies.

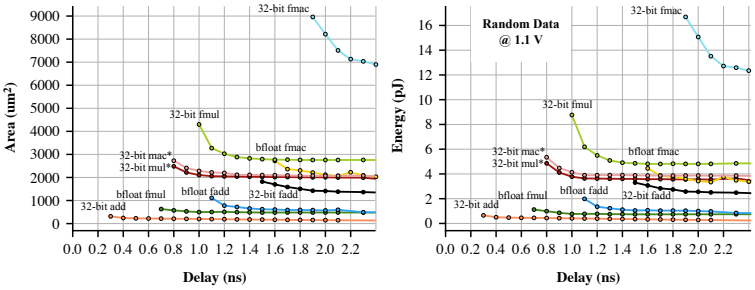
In summary, many architecture students struggle to think from first principles about tradeoffs in performance and energy. There is a need for a first-order energy modeling approach that can be integrated into lectures to supplement existing teaching methods.

3 Classroom Mechanisms for Energy

We describe a specific approach and course methodology, but there are many ways to develop course materials based on the idea that pipeline diagrams can express both performance and energy. This section will first introduce our approach to building student intuition for reasoning about energy. We then illustrate examples of energy maps, which are the primary first-order tool for building intuition about energy alongside the pipeline diagram. We provide example pipeline diagrams for various microarchitectures and illustrate how power-performance plots can be created using quantitative numbers from this tooling.

3.1 Building Intuition about Energy

Because students have different backgrounds upon entering a computer architecture class, we build intuition for energy from fundamental equations. Figure 2 shows a series of figures that bridge between what is taught in introductory physics classes (i.e., $E = \frac{1}{2}CV^2$) and complex hardware units with per-access energy costs (e.g., a processor or other SoC-level block). Figure 2(a) shows the fundamental energy equation representing an ideal capacitor charging and discharging, and Figure 2(b) shows how an entire processor can be "accessed" in order to execute an instruction with some energy cost, corresponding to a single, fixed, equivalent capacitance of $C_{eq,core}$ charging and discharging. Note that if we pause at this abstraction and do not build any further (e.g., suppose that an instruction always costs 2 pJ regardless of which instruction it is), then we have created a simple **instruction-based energy model** that can be surprisingly useful in introductory lectures when discussing simple in-order pipelines. For example in Figure 1 assuming 2 pJ per instruction, one transaction costs 2 pJ and four transactions costs 8 pJ, making it clear how energy increases down the axis. Furthermore, many pipeline effects can be quantified. For example, a branch misprediction would fetch instructions and then squash them, representing wasted work and still adding to the total energy cost. Different aspect ratios in a pipeline diagram can represent lower-power shapes (e.g., few instructions that stall for many cycles, wide and short) and higher-power shapes (e.g., many instructions that complete quickly back-to-back, tall and narrow). Instruction-based energy models tend to model simple pipelines fairly consistently because most instructions access similar hardware resources. However, in more complex systems including out-of-order cores



Operator in 45nm	Energy (Post-Layout) Random Data @ 1.1 V	Energy (Post-Layout) Small Data @ 1.1 V	Energy (Scaled) Random Data @ 0.9 V	Energy (Scaled) Small Data @ 0.9 V	Horowitz ISSCC 2014 @ 0.9 V
8-bit add	0.06 pJ	0.02 pJ	0.04 pJ	0.01 pJ	0.03 pJ
32-bit add	0.28 pJ	0.10 pJ	0.19 pJ	0.07 pJ	0.10 pJ
8-bit mul	0.45 pJ	0.05 pJ	0.30 pJ	0.03 pJ	0.20 pJ
32-bit mul	7.70 pJ	1.89 pJ	5.15 pJ	1.27 pJ	3.10 pJ
32-bit mul*	3.57 pJ	2.00 pJ	2.39 pJ	1.34 pJ	3.10 pJ*
16-bit fadd	1.00 pJ	0.91 pJ	0.67 pJ	0.61 pJ	0.40 pJ
32-bit fadd	2.56 pJ	2.47 pJ	1.71 pJ	1.65 pJ	0.90 pJ
16-bit fmul	0.74 pJ	0.70 pJ	0.49 pJ	0.47 pJ	1.10 pJ
32-bit fmul	4.81 pJ	4.75 pJ	3.22 pJ	3.18 pJ	3.70 pJ

Figure 4: Area, Energy, and Delay of Various Operators in 45 nm Technology — Numbers are collected post-layout for both area and energy in a commercial ASIC toolflow by sweeping target clock period across a large range. Each curve only shows the data points when operators meet timing. Table energy numbers correspond to the "knee" of the curve and reflect a semi-aggressive timing constraint. *Random Data* = uniform random bits toggling; *Small Data* = restricts the data range to [-0.5, 0.5] for floating point and uniform random in the lowest quarter of bits for integer; *Scaled* = energy scaling 1.1 V to 0.9 V with $\frac{1}{2}\ CV^2$ to compare with [8]; *Horowitz Data* = [8]; (*) = truncated result of lower 32 bits for 32-bit mul, unclear whether this was assumed in [8].

and hardware accelerators, transactions can take different paths and resources may have more complex interactions.

We move one level deeper to pipeline diagram letters and a **pipeline-stage-based energy model**. Figure 2(c) shows how we can break the single, monolithic $C_{eq,core}$ into multiple equivalent capacitances. For example, in a simple traditional five-stage pipeline we might have $C_{eq,F}$, $C_{eq,D}$, $C_{eq,X}$, $C_{eq,M}$, and $C_{eq,W}$ corresponding to five different per-access energy costs. An instruction that flows through the pipeline would access each stage and accrue each incremental energy cost. A similar understanding can be applied to other hardware designs, including a pipeline that supports out-of-order execution with multiple execution pipes (e.g., Y-stages for multiplier pipe, L-stages for load pipe, S-stages for store pipe, I-stage for issue to functional units, C-stage for commit stage, etc.). A pipeline-stage-based energy model allows transactions that take different paths through the hardware to have different energy costs. The method extends to non-processor pipelines as well including pipeline stages in multi-hop interconnection networks and pipelined caches. This narrative leads us to the concept of an energy map which simply gathers these ideas into one centralized place.

3.2 Energy Maps

Energy maps are the key teaching construct in our approach. The energy map is a centralized construct that gathers all of the pipeline-stage-based energy models for a given hardware design into one place. More concretely, it is the set of all unique pipeline diagram letters as well as their mappings to per-access energy costs. Figure 3 illustrates an energy map for a simple, five-stage pipeline. The instructor provides the base assumptions, and students fill in the per-access energy costs for each pipeline diagram letter. In this section, we discuss both of these two parts in detail.

3.2.1 Base Assumptions The primary role of these numbers is to abstract the technology for an audience of computer architecture students. These numbers should reflect reasonable trends in a modern technology. They should be abstract enough to shift concern away from the details of the VLSI implementation (e.g., cell placement density, wire congestion, metal stacks).

We capture two major categories of hardware that students can consider as building blocks: combinational logic (e.g., ALU and multiplier logic) and table-based lookup (e.g., register files and SRAMs).

Energy Map Worksheet

	Fairly Accurate		Less Accurate
1.	`mul`	1.	`lw`
2.	`addi`	2.	`lw`
3.	`mul`	3.	`lw`
4.	`addi`	4.	`lw`
5.	`mul`	5.	`lw`

Explain why in one sentence (for each)

Figure 5: Reasoning with Energy Maps — Craft a sequence of five instructions where the energy estimate with your energy map will be fairly accurate. Craft another sequence that will be far less accurate given the assumptions made in your energy map.

Figure 3 shows how the instructor provides arithmetic logic energy numbers with E_{alu} and E_{mul} as well as table-based lookup energy numbers with E_{rf} and $E_{\$}$. After being given a reference within these two categories, students can reason about the energy costs of other logic in terms of these base assumptions. For example, the five-stage pipeline supports full bypassing from the X-stage, M-stage, and W-stage to the D-stage. A student may reason that the bypass network within the decode stage is of similar complexity and in the same category as the ALU. They may then estimate $1 \times E_{alu}$, or one ALU access worth of energy cost, to approximate the energy cost of using the bypass network in the D-stage.

It can be challenging for instructors to find reasonable base assumption energy numbers. We provide pointers to literature and also supplement our own studies to help shed light on how these numbers can be consistently produced for different technologies and future lectures. Horowitz [8] provides quantified absolute numbers to estimate instruction energy breakdowns in a 45 nm technology. We illustrate our own study of area, energy, and delay in the same 45 nm technology in Figure 4, using post-layout numbers produced in a commercial ASIC toolflow. Each curve represents an operator (e.g., a 32-bit floating-point operation) and plots points for different final post-layout designs produced at the given target clock periods. The y-axes show the best area and energy achieved by the commercial tools for each target clock period.

The data shows how picking a single number for each operator is non-trivial. First, an entire curve of data exists just for sweeping the target clock period. Note how area and energy rapidly increase when an operator targets a higher frequency (e.g., `32-bit fmul` increases from roughly 5 pJ per access to 9 pJ per access when pushed from 750 MHz to 1.0 GHz). Also, energy fundamentally depends on the input data. The table numbers in Figure 4 show how uniform

random toggling of the data field bits (i.e., "Random Data") produces much higher energies for each operator compared to using low-magnitude data that is skewed towards zero (i.e., "Small Data").

In summary, the table data shown in Figure 4 roughly agrees with the literature data in Horowitz [8] assuming low-magnitude data with semi-aggressive timing constraints. We hope this study helps shed light on how such numbers can be produced from a commercial ASIC toolflow and a given technology.

3.2.2 Assigning Per-Access Energy Numbers Figure 3 also demonstrates how students can reason about energy by building from their own understanding of the events within each pipeline stage. The key intuition is that **course lectures teach students about the microarchitecture, and students use that knowledge to build their energy maps**. For example, students know that the writeback stage (W) writes a value to the destination in the register file. Students can estimate that this charges $1 \times E_{rf}$ of energy cost. Energy maps may encode specific assumptions. For example, the execute stage (X) computes an arithmetic operation, but students may reason that not every instruction is an ALU or a multiply operation and therefore assign to the X-stage the average of the two energies for accessing the ALU and the multiplier, reflecting a scenario in which half of the instructions access the ALU and the other half access the multiplier.

Our approach firmly adheres to the mantra that *"all models are inaccurate, but some are useful"*. The pipeline-stage-based energy map by definition does not have enough detail to be highly accurate. We must assign a single number to represent the many ways to access a given pipeline stage. We will see in the next section, however, that high-level trends produced by energy maps to compare performance and energy are remarkably similar to the intuition an experienced architect would expect.

Because our goal is to teach students to reason about energy, it is less important for the energy maps to be highly accurate. It is far more important for students to instead realize why their energy maps may be inaccurate. Figure 5 shows an in-class activity worksheet in which students can reason about the quality of their energy maps. Students are asked to produce a sequence of five instructions for which their energy estimate would be fairly accurate, and another sequence where the energy estimate would be less accurate. For example, every X-stage letter in Figure 3 assumes that instructions access either the ALU or the multiplier (with 50% probability). A less accurate instruction sequence could be a sequence of five `nop` instructions, which access neither unit, or perhaps a sequence of five load-word instructions, since the energy map assumes that only 20% of the instructions are memory-related.

3.3 Evaluation with Power-Performance Plots

By combining pipeline diagrams with energy maps, students are equipped to compare the performance, energy, and power of different microarchitectures. Figure 6 shows how we can compare a single-issue, in-order core to a four-way superscalar, out-of-order core with an issue queue, reorder buffer, and register renaming mechanisms. The results show a 1.4× speedup by trading away 4.9× power, which reflects the general intuition of experienced architects for the tradeoffs of these two types of cores.

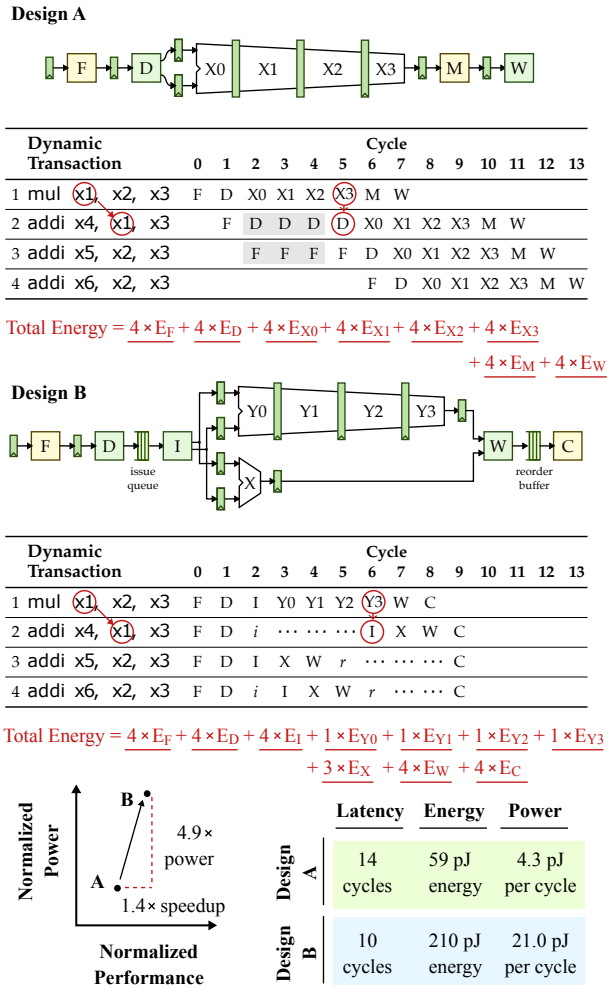


Figure 6: Pipeline Diagram Plots — Power-performance plots for an in-order, eight-stage pipeline processor and a four-way superscalar, out-of-order processor. **New pipeline diagram letters present in Design B**: i = waiting in issue queue (no energy cost), r = waiting in reorder buffer (no energy cost), I = issue stage, Yn = multiplier pipe stages, C = commit stage. **Energy map for Design A**: E_F = 5.0 pJ (fetch); E_D = 4.7 pJ (count $1.5 \times E_{rf}$ to read 1–2 srcs, $1.0 \times E_{alu}$ for bypass network); E_{X0} to E_{X3} = evenly divide 1.1 pJ (assume half multiplies and half ALU ops); E_M = 1.0 pJ (assume 20% of instructions are memory ops); E_W = 3.0 pJ (write to register file). **Energy map for Design B**: E_F = 5.0 pJ (fetch); E_D = 12.4 pJ (count $1.0 \times E_{rf}$ for rename table access, $1.0 \times E_{rf}$ for ROB access, $2.0 \times E_{rf}$ for issue queue access, $2.0 \times E_{alu}$ for combinational structural hazard-checking logic); E_I = 15.6 pJ (count $2.0 \times E_{rf}$ for issue queue access, $1.0 \times E_{rf}$ for scoreboard access, $2.0 \times E_{rf}$ for physical register file access, $3.0 \times E_{alu}$ for combinational dependency-checking logic); E_W = 6.6 pJ (count $2.0 \times E_{rf}$ for physical register file access, $1.0 \times E_{alu}$ for combinational issue queue logic, $1.0 \times E_{alu}$ for combinational ROB update logic, $1.0 \times E_{alu}$ for extra combinational logic); E_C = 12.2 pJ (count $1.0 \times E_{rf}$ for architectural register file access, $2.0 \times E_{rf}$ for physical register file access, $1.0 \times E_{rf}$ for ROB access, $1.0 \times E_{alu}$ for extra combinational logic); E_X = 0.2 pJ (ALU energy); E_{Y0} to E_{Y3} = evenly divide 2.0 pJ (multiplier energy). Please tweak these numbers to correspond to what is taught in the lectures.

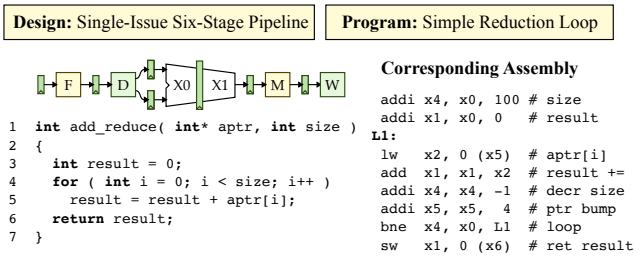
Design A is the classic, five-stage pipeline, but with the X-stage pipelined with four stages (i.e., X0, X1, X2, and X3 stages). The pipeline supports the `addi` and `mul` instructions and other ALU instructions. Regardless of which operation is being computed, all instructions must advance through the full pipeline. The energy map is similar to the one shown in Figure 3, but with the X-stage energy distributed over the four stages. **Design B** is a *quad-issue* superscalar core, and hence we draw four F-stages executing simultaneously in the same column. The pipeline runs at the same clock frequency as in the previous design. The load and store pipes are not drawn but would be set in parallel with the X-pipe (ALU) and Y-pipe (multiplier). Additional structures are drawn including the issue queue (i.e., Tomasulo-style), reorder buffer, and physical and architectural register files with a commit stage. Note that the energy maps for both microarchitectures assume some details (e.g., number of ROB entries) described in lecture, but left out in this paper for brevity.

The final power-performance plot in Figure 6 shows how the pipeline letters can be counted and the key metrics quantified in a table (and plotted on a whiteboard) to compare the two designs. The second design is far more expensive in energy cost compared to the relative gains in performance. Despite the high level of abstraction of energy maps, the relative trends they enable students to visualize tend to be surprisingly reliable. Such plots can be extended to compare many other microarchitectures in a similar way, as long as there is a pipeline diagram present. In summary, students learn from lectures about the hardware structures that are present in each pipeline stage. They then apply their knowledge to build an energy map accompanying the pipeline diagram. This approach can enable evaluating simple in-order cores, complex out-of-order cores, a ring network, or even a multicore system.

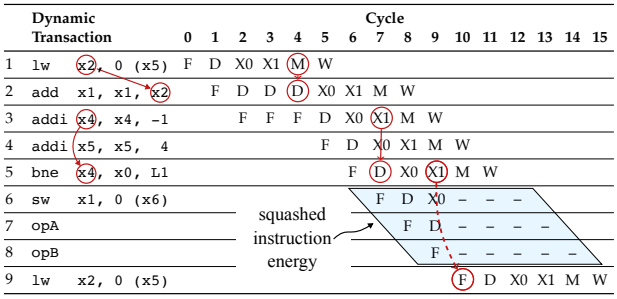
3.4 Grading Approach

There are two observations that set the grading approach for this methodology apart from others. First, there is no authoritative "correct" answer when building an energy map. Many different numerical answers can be similarly correct, as we previously suggested in Figure 5, as long as the strengths and weaknesses of the assumptions made are acknowledged by the student. Second, this methodology is designed to train students to think critically about energy costs and tradeoff evaluation, and the grading methodology should therefore reward critical thinking. For example, if a student describes why a design decision will reduce performance, he or she should receive a similar grade as another student who chooses to discuss a design decision that improves design metrics instead.

We suggest an approach where **grades primarily reward students for taking a position**, arguing from the evidence they collect that their stance is reasonable. These evaluations can be several sentences long (e.g., one sentence to take a position, one sentence in support of the position, and one sentence why the position may be weaker than expected). In particular with this approach, a student can make some mistakes in their pipeline diagrams and energy map, and this is not to be penalized (much). However, they must still use that data to support a position. After all, engineers in both academia and industry frequently make small mistakes in their models while still delivering significant value in their analyses.



(a) Scenario 1 – Execution without dedicated branch prediction hardware



(b) Scenario 2 – Execution with branch predictors in F-stage and D-stage

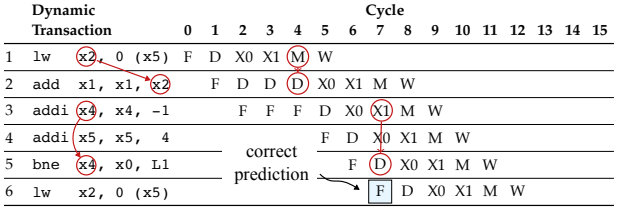


Figure 7: An Exam Question for In-Order Processor Energy with Branch Prediction — A single-issue six-stage pipeline executes a simple reduction loop, with the corresponding assembly code as shown. Under which scenario is the energy higher: with no dedicated branch prediction hardware? or with branch predictors in F-stage and D-stage? Assume that branches resolve in X1-stage.

4 Practice Exam Question

We applied our methodology to introduce energy-driven discussion into EE 557 Computer Systems Architecture at the University of Southern California. This is a graduate course and is taken primarily by students who have already had exposure to computer systems organization. The topics covered in lecture materials start with basic pipelined processors and quickly advance through advanced concepts including superscalar issue, out-of-order execution, register renaming, branch prediction, speculative execution, memory disambiguation, SIMD and vector processors, VLIW processors, vertical and simultaneous multithreading, interconnection networks, and multicore systems. Although we chose an advanced class, there is no reason why these techniques could not be applied in more introductory courses as well.

Figure 7 shows an exam question that explores energy in a single-issue, six-stage in-order pipeline with and without branch predictors. The energy map for this design without dedicated branch prediction hardware is similar to that shown in Figure 3, but with the X-stage energy distributed over the two stages. The pipeline

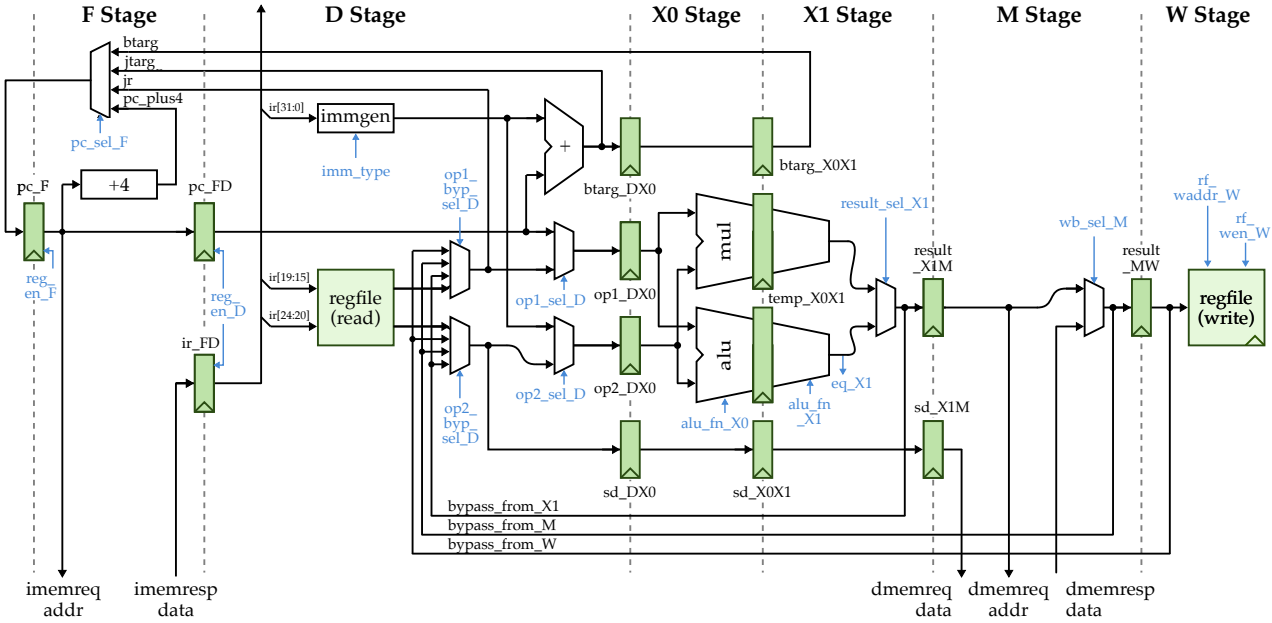


Figure 8: Datapath for Single-Issue, Six-Stage Pipelined Processor

splits the X-stage into X0 and X1 stages, and *branches resolve in the X1-stage*. The detailed datapath diagram is shown in Figure 8.

The core runs a program with a simple reduction loop that sums an array of integers and returns the result. The RISC-V assembly has the main loop at label `L1`. We consider two scenarios executing the loop without dedicated branch prediction hardware (Scenario 1) and with branch prediction hardware in F-stage and D-stage (Scenario 2). Students are taught about classic two-level branch predictors prior to the exam, including how the F-stage can use early branch prediction hardware to predict both the branch target and the branch direction, while the D-stage branch predictors correct and update the F-stage predictors afterwards.

Scenario 1 in Figure 7(a) shows how the branch is predicted not taken, resulting in a misprediction and three fetches to the store-word, opA, and opB instructions. These instructions partially execute before being squashed. The branch (i.e., branch-not-equal, `bne`) resolves in X1-stage. **In Scenario 1, students use their energy maps to quantify the energy lost to squashed and wasted work**. Note that the F-stage and D-stage have high energy costs and that these costs are front-loaded. The energy is already consumed by the time the squash signal arrives. The total energy lost can be expressed as: $3 \times E_F + 2 \times E_D + 1 \times E_{X0}$.

Scenario 2 in Figure 7(b) shows how the branch predictors in F-stage and D-stage correctly predict the backwards loop, and there is no wasted energy from mispredictions and squashed instructions. However, this comes at a cost. **In Scenario 2, students realize that the energies to access the F-stage and D-stage increase due to the branch predictors**. These letters in the pipeline diagram become more expensive. At this point, it may not be clear how the branch predictors affected the total energy. Which effect dominates: the more expensive F-stage and D-stage? or the energy saved by avoiding fetching and squashing instructions?

The exam question indicates that students should take a position of the form: "Scenario 2 with branch predictors *increases* / *decreases* the total energy of this assembly loop". They then support their position with quantitative evidence from their pipeline diagrams (e.g., counting letters) and energy maps (e.g., estimating how much more energy the branch predictors add to the F-stage and D-stage). After computing the squashed instruction energy in Figure 7(a), they can take a firm position on whether the total energy cost of the predictors will exceed the energy saved. Note that students may be allowed to draw their own branch prediction hardware which may have different energy costs, unless this detail is provided.

To clarify the intuition behind this exam question, we note that sophisticated branch predictors are often added to high-performance out-of-order cores with deep pipelines, which cannot maintain high performance without extremely high accuracy in their branch predictors. There is a tremendous performance and energy cost for squashing tens or hundreds of in-flight instructions in such cores, which justifies the cost of the branch prediction logic. However, the exam question assumes a simple six-stage in-order core, and the performance and energy lost to squashed instructions may be quite small. From the perspective of energy alone as asked in this exam question, the costs of implementing a complex, sophisticated branch predictor will likely outweigh the energy to fetch and squash a few instructions. However, a simple branch predictor of low complexity may still represent a worthwhile tradeoff.

5 Conclusion

Computer architecture students today have little access to well-developed first-order tools for reasoning about energy. On the other hand, VLSI students learn about energy but do not connect these intuitions back up to the architecture level. We have designed an approach that builds energy modeling directly into pipeline

diagrams, providing a way to extend energy-driven discussion into existing course materials which already frequently use pipeline diagrams to discuss performance. From our course offering, we notice that students are more consistently trained in how to reason quantitatively about the performance *and* energy implications of hardware design principles and computer architecture concepts. Given that semiconductors are becoming more important from a workforce and education perspective, the implications of tying these vertically integrated themes directly into a lecture can be significant. We hope that other computer systems lecturers can glean useful lessons from our work and develop materials to train the next generation of computer architects to think from first principles about performance and energy.

References

[1] Synopsys Reference Methodologies. http://solvnet.synopsys.com/rmgen.

[2] V. Akhlaghi, A. Yazdanbakhsh, K. Samadi, R. K. Gupta, and H. Esmaeilzadeh. SnaPEA: Predictive Early Activation for Reducing Computation in Deep Convolutional Neural Networks. *Int'l Symp. on Computer Architecture (ISCA)*, Jun 2018.

[3] N. L. Binkert, B. M. Beckmann, G. Black, S. K. Reinhardt, A. G. Saidi, A. Basu, J. Hestness, D. Hower, T. Krishna, S. Sardashti, R. Sen, K. L. Sewell, M. S. Altaf, N. Vaish, M. D. Hill, and D. A. Wood. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, Aug 2011.

[4] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. *ACM SIGARCH Computer Architecture News*, May 2000.

[5] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE Journal of Solid-State Circuits (JSSC)*, Jan 2017.

[6] M. Dubois, M. Annavaram, and P. Stenström. Parallel computer organization and design. *Cambridge University Press*, 2012.

[7] J. L. Hennessy and D. A. Patterson. Computer architecture: a quantitative approach, 6th edition. *Elsevier*, 2018.

[8] M. Horowitz. Computing's energy problem (and what we can do about it). *Int'l Solid-State Circuits Conf. (ISSCC)*, 2014.

[9] N. P. Jouppi, D. H. Yoon, M. Ashcraft, M. Gottscho, T. B. Jablin, G. Kurian, J. Laudon, S. Li, P. Ma, X. Ma, T. Norrie, N. Patil, S. Prasad, C. Young, Z. Zhou, and D. Patterson. Ten Lessons From Three Generations Shaped Google's TPUv4i. *Int'l Symp. on Computer Architecture (ISCA)*, 2021.

[10] K. Koul, J. Melchert, K. Sreedhar, L. Truong, G. Nyengele, K. Zhang, Q. Liu, J. Setter, P.-H. Chen, Y. Mei, M. Strange, R. Daly, C. Donovick, A. Carsello, T. Kong, K. Feng, D. Huff, A. Nayak, R. Setaluri, J. Thomas, N. Bhagdikar, D. Durst, Z. Myers, N. Tsiskaridze, S. Richardson, R. Bahr, K. Fatahalian, P. Hanrahan, C. Barrett, M. Horowitz, C. Torng, F. Kjolstad, and P. Raina. AHA: An Agile Approach to the Design of Coarse-Grained Reconfigurable Accelerators and Compilers. *IEEE Trans. on Embedded Computing Systems (TECS)*, Jan 2023.

[11] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. *Int'l Symp. on Microarchitecture (MICRO)*, Dec 2009.

[12] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi. CACTI 6.0: A tool to model large caches. *Int'l Symp. on Microarchitecture (MICRO)*, Dec 2007.

[13] Y. S. Shao, B. Reagen, G.-Y. Wei, and D. Brooks. Aladdin: A pre-rtl, power-performance accelerator simulator enabling large design space exploration of customized architectures. *Int'l Symp. on Computer Architecture (ISCA)*, Jun 2014.

[14] C. Torng, P. Pan, Y. Ou, C. Tan, and C. Batten. Ultra-Elastic CGRAs for Irregular Loop Specialization. *Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb 2021.

[15] C. Torng, M. Wang, and C. Batten. Asymmetry-Aware Work-Stealing Runtimes. *Int'l Symp. on Computer Architecture (ISCA)*, Jun 2016.

[16] Y. N. Wu, J. S. Emer, and V. Sze. Accelergy: An architecture-level energy estimation methodology for accelerator designs. *Int'l Conf. on Computer-Aided Design (ICCAD)*, Nov 2019.