Project Based Learning in Computer Architecture

Neeraj Goel and Venkata Kalyan Tavva
Department of Computer Science and Engineering,
Indian Institute of Technology Ropar, Ropar, India.
neeraj@iitrpr.ac.in, kalyantv@iitrpr.ac.in

Abstract

Computer architecture and organization is one of the core undergraduate courses in computer science and engineering discipline. The course is usually taught in the second year of the curriculum and primarily focuses on learning the intricate details of an Instruction Set Architecture (ISA) and design of a processor system. Many times the laboratory component in this course is usually limited to writing assembly programs. Some universities use different types of simulators to enhance student learning in computer architecture. Nikolic et al. [1] compare different type of simulators that were used in various computer architecture courses. Most of these simulators help in demonstrating a particular aspect of the architecture. For example, cache simulator helps to understand the effect of different memory access pattern on performance, while some simulator can help visualize the pipeline.

A course in computer architecture primarily has four modules, (a) machine instructions, (b) instruction execution and pipeline design, (c) memory technologies and hierarchy, and lastly, (d) input-output devices and interfaces. Our objective is to design a lab component for this course that can enhance understanding of all these modules. During the first few weeks of the course, for a given ISA (typically a RISC type) students are given lab assignments on assembly programming. Next, we focus on student projects that improve their understanding of modules (b), (c), and (d). The concepts of these modules are best learned when the students design their own processor architecture in sufficient detail. Accordingly, students need to build a cycle accurate modelling of a processor system in a high level language such as `C, C++, Java` to design their own processor simulator. A big challenge for the students (in their third/fourth semester) in designing such a simulator is their limited exposure in writing and handling large software. To address this hurdle, we provide students with a detailed design document of the expected software. Along with the detailed design document, a base code template is provided that gives an idea on the overall flow of the design. This approach helps students to jump start and finish the project in the stipulated time.

Additionally, the objective of this simulator is not on the performance, but on accurately modelling various architectural features taught in the class. For example, all the signals along with their bit-width are supposed to be modelled, concept of encapsulation needs to be kept in mind, in case a particular signal is not accessible by a hardware unit, the variable corresponding to that signal can be not be accessed by model of that hardware unit, handshake among different units, etc.,

With this proposed approach, students can appreciate the complexities involved in designing every hardware unit and concepts related to its interface with other units. Next section discusses the proposed project approach in more detail.

Designing simulator of a processor

The overall project of designing simulator of a processor is divided into multiple phases, with some of the modules being optional. These phases and modules are briefly described next. The reader is referred to [2], [3] for any of the terms used below.

Phase I: Modelling single cycle processor

The objective is to model a single cycle processor in a high-level language. Considering an ISA or subset of instructions from it, the students need to understand and build a functional model of instruction execution. This task primarily works as a base for the next phase of the project. Input to this module is a text file containing input program written in hex or binary format. The simulator takes this file as an input and as the first task, uses it to initialize memory of the processor. The simulator models different stages of a processor as different functions with a clear interface and different resources as variables. The expected output of this module is correct register values and correct memory state at the end of execution of each instruction. Also, few statistics like the total number of cycles, instructions, Cycles Per Instruction, etc., can also be part of the output.

Phase II: Modelling Pipeline Processor

The objective of this phase is to extend Phase-I to include pipelined execution of instructions. Students will, therefore, model pipeline aspects like data dependencies, data-forwarding, stalling, and control hazards. Given that this phase of the project is built after the single cycle processor model (also by using the previous code), students will understand the hardware implications of pipelining. All the pipeline registers are modelled to emulate cycle accurate model of the processor pipeline.

Expected output along with the proper functioning of all the above-mentioned features, is statistics like total number of instructions, the total number of cycles, CPI (cycles per instruction), number of hazards (data, control), number of data forwards, number of branch mispredictions, etc., Such a self-designed model can help the students understand and appreciate impact of various features.

Phase III: Integrating Cache Model

The primary objective of this phase is to understand the role and working of cache memory along with its interface with the processor. In this task, the students require a good understanding of the cache controllers in addition to the basic working of the data and tag arrays in caches. Students build a modular cache structure, wherein, the end user can specify the cache-type (Direct-mapped/Set-associative/Fully-associative), cache-size, block-size, replacement policy (LRU/MRU/FIFO/Random), etc. The cache model needs work with the pipeline structure in a lock-step fashion. One of the possible output of the cache model can be statistics like the total number of accesses, hits, misses (cold, conflict and capacity). Such a comprehensive task will help students understand the impact (using the statistics) of various design parameters in a cache for a given application.

Such a simple cache model can further be extended to include more modules like split caches, multiple cache levels, Miss Status Handling Registers, Prefetching logic, to name a few.

Optional Modules

1. Building an Assembler for an ISA. Since the input to the processor simulator is binary program image and not the assembly code, students can be directed to build a simple two-pass assembler, that will produce a binary file as output in the desired format. This module will help students understand the role of instruction encoding, various instruction fields, and addressing modes of an ISA. Besides, students will also understand the need for assembler directives, methods to handle instruction labels, control instructions, and offsets, pseudo instructions, etc.

2. Adding a custom instruction. Since most of the basic instructions were three-operand instructions, where there are two reads from, and one write to the register file, students were asked to implement additional instructions such as *mac*, which also have three operands, however, require three reads and one write. Another example is the inclusion of a *swap* instruction, that requires two reads and two writes to the register file.

3. Multiple issue processor. The basic pipeline design needs to be modular to work as a multiple issue VLIW processor. Input is a correct assembly program with an assumption that the programmer will write independent instructions in-pairs. In case an independent instruction is not available, NOP instruction will be appropriately included in the assembly program. Pipeline processor will be extended to handle data-dependency and control dependent model to multiple issue scenario.

4. Multiprocessor. A simple multiple processors can be designed by having multiple instances of the single processor simulator. However, processor ISA needs to be extended to support synchronization across multiple processes. Further, additional instructions are required to identify the processor-id. Using these extensions, students can model a bare minimum multi-processor system that works with shared memory. Since the processor is tested using a multi-threaded application written in assembly, students get an opportunity to learn low level synchronization mechanisms.

5. Integrating Interrupt and Exceptions controller. Exceptions are part of any computing system, handling them in a simple simulator requires hardware support of storing registers onto a *stack*, jump to its subroutine. Further actions are taken by the subroutine. Implementing an exception in simulator exposes students to the minute details of the exception controller. The same method is extended to support interrupts.

6. Branch Prediction module. Students can also extend the basic pipeline of Phase-II with features like static/dynamic branch prediction. Impact of simple static (always taken/not-taken, forward taken, backward not-taken) and dynamic (1-bit/2-bit/tournament) prediction can be studied.

Reflections from project based-approach

This approach has been used by instructors over the past four years. Three different architectures have been modelled by students in different years: ARM, RISC-V [3], and simpleRISC [2]. The projects done by the students were a subset of possibilities mentioned in the previous section. All the possible projects mentioned in the previous section have been implemented by at least one set of students. Group and individual interview with students show their improved learning of the fore-mentioned concepts.

An explicit question to students on their learning experience from the project was asked, to which, the instructors received extremely positive feedback. Many of the students not only appreciated the technical challenges of the project but also expressed improved confidence in their overall coding, teamwork, and software version management skills to name a few.

References

[1] Bosko Nikolic, Zaharije Radivojevic, Jovan Djordjevic, and Veljko Milutinovic, "A survey and evaluation of simulators suitable for teaching courses in computer architecture and organization," *IEEE Transactions on Education*, vol. 52, no. 4, pp. 449–458, 2009.

[2] Smruti R Sarangi, *Computer Organisation Architecture*, McGraw-Hill Education, 2015.

[3] David A. Patterson and John L. Hennessy, *Computer Organization and Design RISC-V Edition: The Hardware Software Interface*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2017.