AC 2008-2818: A PROCESSOR DESIGN PROJECT FOR A FIRST COURSE IN COMPUTER ORGANIZATION

Michael Black, American University

A Processor Design Project for a First Course in Computer Organization

Abstract

Although many of today's students are savvy computer users, paradoxically they often find computer design abstract and difficult to visualize. To make the material more tangible, we have developed a novel three part term project that requires students to develop and simulate their own processor. Students work in teams to devise and encode their own instruction set, design a datapath and microcontrol instructions to execute their instruction set, and simulate a model of their processor on a computer.

While similar CPU design projects have been offered at other institutions, this project is unique for three reasons. First, it gives students freedom to be creative by letting them design their own instruction sets, while constraining them to practical limits by requiring them to encode their instruction set and compile a small program for it. Second, by building a simulator for their processor, students can actually see their processor run a program. Third, the project needs no special hardware or software, and does not require the students to have any extensive background experience either in digital logic design or programming. It can consequently be offered in programs that do not have access to specialized equipment, and is suitable for students across diverse disciplines.

This project has been given to three classes of electrical engineering students and two classes of computer science students at different universities. Qualitative feedback was solicited from each class. Students have universally expressed that this project has made the subject matter feel more tangible to them and has greatly increased their enjoyment of the course. Students have tended to take advantage of the design freedom given to them by experimenting with unusual instruction sets, and in one case a team actually built their processor in hardware.

This paper describes the project itself in depth. It further discusses the student populations on which the project has been tried and the course contexts in which it was given. Finally, the paper looks at the student response to the project and the creativity of their submissions.

1. Introduction

Many students have difficulty when encountering computer organization for the first time. This is due partly to the unfamiliarity of the material, but is also because computer architecture as a discipline is different from the fields that students encountered earlier. Unlike calculus, computer design has little underlying theoretical foundation. Instead, from its beginnings, computer design has always been a process of trial and error, with modern computers being designed the way they are because "it works best", rather than any deeper theoretical reason. Students encountering the field for the first time need to understand not just how computers

work, but why they work that way, and what the process was that convinced architects to build computers the way they do.

In this paper we describe a CPU design project for a first course in computer organization. The project consists of three components: designing an original instruction set, devising a datapath and control unit, and writing a simulator for their processor. Students completing this project learn not only how processors work, but also experience the tradeoffs and constraints involved in designing a processor. Students also learn to evaluate their processor in a similar way to how processors are evaluated in actual computer architecture research. This project has been assigned to electrical engineering students at the University of Maryland, College Park for three semesters, and to computer science students at American University for two semesters.

Processor design projects are included in computer organization classes at several other universities. Courses that require students to modify existing processor simulators are fairly common [2,3]. Courses at Rose-Hulman and Cornell [4,5] have students design processors using specialized software, such as LogicWorks. Processor design hardware kits have even been produced to allow students to easily implement computer design in hardware [1].

However, this project is unique for several reasons. First, students are given creativity to design their own instruction sets rather than use a preexisting one. Second, unlike similar projects at other universities, the project requires no specialized hardware or software. Third, the project does not require students to know any particular background knowledge before the course apart from basic programming, a typical prerequisite for computer organization courses. Fourth, by requiring students to simulate and evaluate their processors, the project teaches how real processor research and evaluation are performed.

In Section 2 of this paper, we describe the student populations for which this project was written. Section 3 describes the three parts of the project in depth, and tells how it was adapted for electrical engineering and computer science classes. Section 4 gives both qualitative and quantitative student assessment of our project, and describes examples of exceptional or follow-up student work.

2. Student Populations

2.1 Electrical and Computer Engineering Students at the University of Maryland

ENEE350, “Computer Organization”, is a required course for all electrical engineering and computer engineering students at the University of Maryland, a large state university. Students taking the course are expected to have taken prior courses in digital logic design and introduction to programming. Approximately the first five weeks of the course teaches assembly language in MIPS. The second five weeks covers instruction set encoding and datapath design. The remainder of the course typically covers cache and virtual memory.

Our project was assigned to Frostburg State University students for three semesters of the course. The majority of the students were juniors, with a handful of sophomores and seniors. Each class had between 35 and 50 students. The classes also included between 3 and 6 distance students located at Frostburg State University who were taking the class over a television network. All students in the class had access to computers running UNIX. The students in these classes tended to complete the projects in teams of three.

2.2 Computer Science Students at American University

CSC540, "Computer Organization and Design", is a course offered at American University, a primarily liberal arts institution. CSC540 is a required course for all computer science undergraduate students and computer science graduate students who have not had an equivalent course as an undergraduate. Students taking this course are expected to have substantial programming experience consisting of at minimum two courses in computer science, but have not necessarily had any prior exposure to digital logic design. The course briefly covers MIPS assembly, then covers instruction set design, datapath design, and cache. The course also covers pipelined and superscalar processors, and some more modern processor optimizations. CSC540 is generally a more theoretical course than ENEE350.

Our project was assigned to two semesters of American University students. The first class consisted of eight students, who were a mix of juniors, seniors, and masters-level graduate students. The second class consisted of seven students, who were a mix of sophomores, juniors, and seniors. Students had access to PCs running Windows and to a Java IDE. The students in these classes tended to complete the projects individually.

3. The Project

3.1 Project Objectives

When designing the project we had several objectives.

3.1.1 A Working Model

Much like similar successful projects assigned at other schools, we want our students to have a tangible processor when the project is completed. Students should be able to see their processor executing programs.

3.1.2 Creative Design

Processors designed over the decades have differed vastly in instruction set and architecture. We feel that students cannot appreciate the design decisions and tradeoffs in constructing a processor if they are dictated an instruction set to use. The unique aspect of this project, and the most challenging, is leaving the choice of instructions, registers, and addressing modes entirely up to students.

3.1.3 Understanding Microarchitecture

One of the core parts of a computer organization course is understanding how a computer can be formed from logic components. We feel it is vital that students design their own datapath.

The project is divided into three subprojects. There are several reasons for this. First, it forces students to space out their work and not attempt the project at the last minute. Second, since later parts of the projects build off of earlier parts, students are able to get feedback on earlier stages of the project and make corrections in time. Third, later portions of the project depend on the students knowing material that is not typically discussed until partway through the course. Dividing into parts allows students to complete substantial portions of the project before this material is taught.

3.2 Subproject 1: Instruction Set

The first subproject requires the students to design and encode their own instruction set. The project requires students to determine which registers their machine should have, create a sequence of machine instructions, and produce a unique binary encoding for each instruction. The students are restricted to a fixed-length encoding of 8 bits/instruction, and their instructions must be capable of addressing at least 256 bytes of memory. Copying an existing instruction set is not permitted. At minimum, instruction sets must contain instructions that transfer data between registers and memory (if they choose to have registers), basic arithmetic instructions, and conditional control instructions. To ensure that their instruction set is capable of running real programs, students are given a small for loop in C or Java that adds a series of numbers together and stores the result in memory. Students are required to translate this program into their own assembly code and machine code and submit that with their instruction set. The first subproject is submitted in paper form as a report; students are required to explain and justify their choices of instructions and registers. Figure 1 shows the instruction sets submitted by two students for this subproject.

When beginning the first subproject, students have written simple programs in an assembly language (usually MIPS), and have had at least one lecture on encoding instructions. Students are typically given two weeks to complete this first subproject, which is usually ample time. The second subproject is not usually assigned for at least another week. This gives time for the submissions to be graded and returned, and for the students to make corrections to their instruction set before beginning the next part. The subproject is primarily graded on whether the instruction set is well chosen and correctly encoded, as well as whether the small program is

compiled correctly. Submissions are also graded on creativity; students earn extra points if their instruction set is substantially unlike any that they studied before, or has any particularly inventive features.

Instruction Set 1

01. ADD - Addition, Add the top two values on the v. stack
02. SUB - Subtraction, Subtract the top two values on the v. stack
03. LB - Load Byte, using the address that is on the operation stack
04. SB - Store Byte, using the address that is on the operation stack
05. JAL - Jump And Link
06. BEQZ - Branch to the memory address that is on the operation stack
07. SBVS - Store Byte on V. Stack,
08. PEEKVS - Peek on V. Stack
09. POPVS - Pop from V. Stack
10. SWAPVS - Swap on V. Stack
11. LRVSP - Load Register - V. Stack Pointer
12. SRVSP - Store Register - V. Stack Pointer
13. LRRA - Load Register - Return Address
14. SRRA - Store Register - Return Address

Instruction Set 2

ADD	*adds first 2 elements on top of stack*
SUB	*subtracts 2nd element from first and places the result on top*
PUSHM	*uses the 1st element on the stack as memory address and loads from memory to the top of stack*
PUSHI IMM	*pushes an immediate on to the top of the stack*
POPM	*uses the 1st element on the stack as memory address and loads the second element to memory from the top of stack*
POP	*deletes the top element*
JEQ IMM	*If the top element is equal to immediate it jumps to the section of code referenced by the 2nd element on stack*
RET	*Returns to area of code specified by the first element in the stack*
COPY	*Makes a copy of the top element on the stack and places it on the stack*
COMB	*Combines the top 2 elements on the stack into one 8 bit immediate*
SWAP	*switches the 2 top elements*

Figure 1: Two Student Instruction Sets

3.3 Subproject 2: Microarchitecture

For the second subproject, students are required to design a datapath and control instructions that will execute their instruction set. By the time this project is assigned, the course has covered block diagrams of datapaths and microassembly control code. The students will have also received their graded instruction set submission and have made the necessary changes. Two weeks is typically given to complete this subproject, and like the first, it is submitted as a report.

Students are required to submit a detailed schematic of their datapath. The datapath must consist solely of registers, including special registers such as the program counter and instruction registers, register files, ALUs, multiplexors, sign extension units, and the wires connecting them together (students are not required to design the ALU). The datapath does not need to show the control state machine, the control input to the multiplexors, or the clock input to the various registers. Figure 2 shows an actual datapath submission. Students are also required to submit their control code. The control code for each instruction must be written in a series of microassembly instructions. For each microassembly instruction, students must specify the state of each multiplexer and device in their system requiring a control input, and must also specify

which registers receive data on that instruction. Although students are not specifically required to design the control state machine, they have provided the necessary information needed to build it.

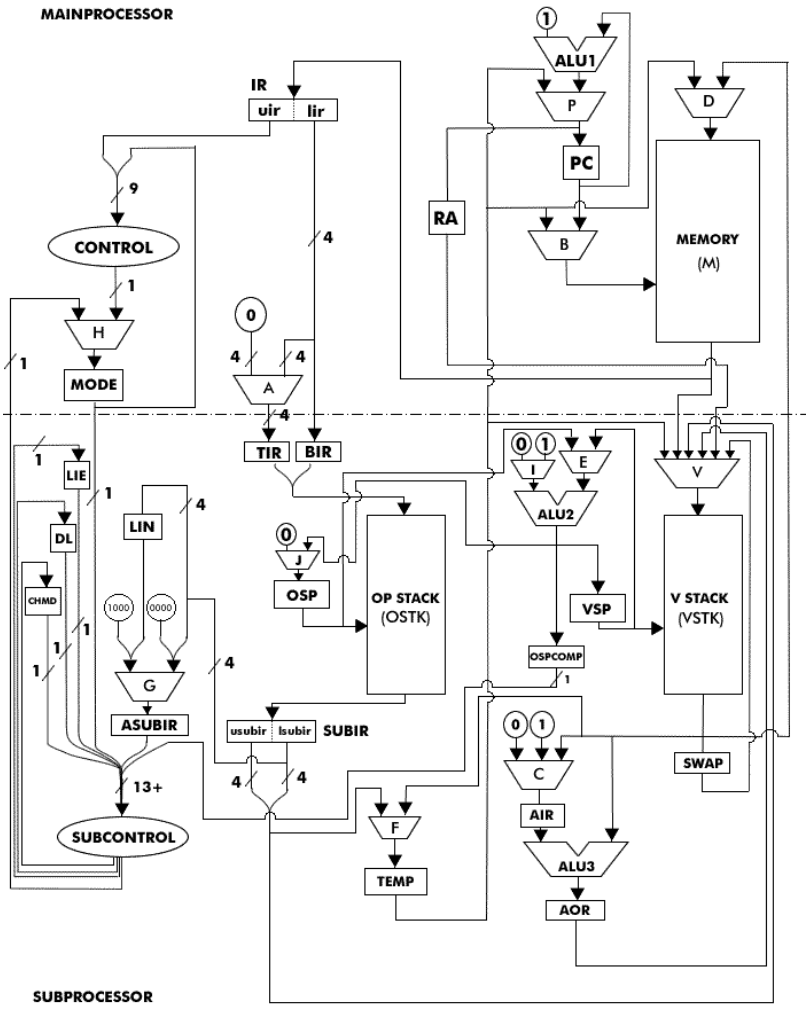

Figure 2: Student Datapath Submission

3.4 Subproject 3: Simulator

The third subproject requires students to program a simulator for their processor, using their control code from the second subproject. To demonstrate that their simulator works correctly, and to study the efficiency of their processor, the project requires students to execute the program from subproject 1 on their simulator with varying data sizes. Students are given two to three weeks to complete this project. When completed, students must submit their simulator code, their simulator output file, and a report discussing the performance of their processor.

The processor simulator is written either in C or in Java, and models the registers, RAM, and other components of the machine. Simulators are required to be cycle-accurate, meaning that each microassembly control instruction must be modeled. Students start their simulation by loading the machine code of the subproject 1 program into the simulated RAM array. When the simulation is concluded, their simulator must write the contents of memory to an output file, allowing students to verify that their program worked correctly. The simulator also prints out the number of instructions executed and the number of simulated cycles, allowing students to evaluate the instruction/cycle performance of their processor. Figure 3 shows a screenshot of a simulator submitted by a student. In this particular example, the student added a GUI, assembler, and step-by-step tracer to his project.

In some classes to which this project is given, many students do not have sufficient programming skills to write a processor simulator from scratch. For these classes, students are provided template simulator code in C. This template contains the code to load a machine code text file into simulated RAM, step through a series of cycles, and write the simulated RAM to an output file. Given this template, students need to add their registers as variables, and their microassembly instructions to the main simulation loop reformatted as C commands. The programming required in this part, once the template is given, has been found to be within the ability of practically every student attempting the project.

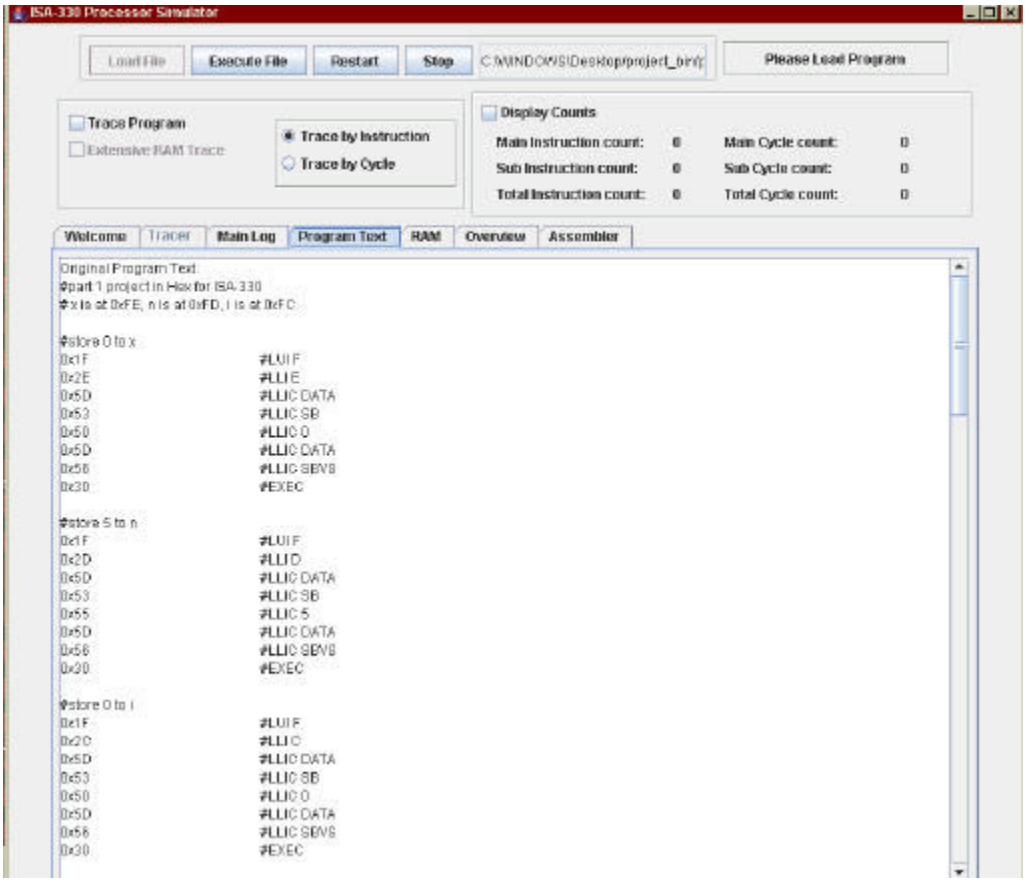

Figure 3: Student Simulator Submission

3.5 Adaptations

Because the background experience of computer science students is different from that of electrical engineering students, the project assigned in CSC540 had some minor differences to that assigned in ENEE350, and the guidance needed on various parts of the project was different for the two groups of students.

The danger of the first subproject is that students might create an instruction set that is not Turing Complete and is incapable of executing some programs. Students consequently need guidance to ensure that their instruction sets are sufficiently comprehensive. The nature of the guidance, however, is different for the different majors. The project assigned to the electrical engineering students included a description of the types of instructions needed in the processor. While no instructions were explicitly named, a detailed description of the types of instructions needed, such as arithmetic instructions or immediate loading instructions, was provided with the project description. With the computer science students, since they have had some formal mathematical training, a different approach was used. They were provided with a sample RiSC instruction set that is Turing Complete. In their subproject 1 report, they were required to use that given instruction set to prove that their instruction set was also Turing Complete.

The second subproject was generally easier for the electrical engineering students than the computer science students, as the electrical engineering students had previously taken a digital circuit laboratory course and had better understanding of how registers and multiplexors work. To compensate for this, the second subproject assigned to the computer science students included a sample datapath and control to handle two of the instructions from the RiSC instruction set; this was unnecessary for the electrical engineering students.

As mentioned before, a simulator template was given to the electrical engineering students for the third subproject. This was unnecessary for the computer science students, who were also consequently given a choice of languages (all chose Java). As the third subproject is principally a programming project, computer science students were also able to go further than the electrical engineering students, and build more elaborate simulators, some including a GUI component or an assembler. In contrast, the electrical engineering students typically submitted more basic simulators, but designed more creative instruction sets.

3.6 Concerns and Solutions

Over repeated offerings of this project, we found that students tended to make similar mistakes. Below are the most common errors and our solutions.

3.6.1 Instruction set is similar to MIPS

Our first concern was that the students' instruction sets would be identical to MIPS. This tended to happen frequently the first time the project was offered. Students would submit instructions

that had the same format and purpose as the equivalent MIPS instructions, in most cases simply renaming the instruction. This problem was largely solved in subsequent classes by giving students examples of diverse instruction sets, including 8080, x86, JVM, PDP-8, and Alpha.

3.6.2 Instruction set is too ambitious

Our finding was that excessively ambitious instruction sets tended to come from the best and worst students in the class. The best students tended to devise complex machines with many layers of dereferencing. Students with poor understanding of course material tended to include instructions that are simple to understand but complex to implement, such as floating point arithmetic or structured programming concepts such as while-repeat, and generally omitted branch instructions. These problems were generally easy to repair. In most cases, the instruction set could be made practical by simply removing instructions, rather than making large changes.

3.6.3 Datapath is unable to execute the instructions

There were only a couple problems that tended to occur in students' datapath designs.

- Misuse of buses to make the datapath excessively simple
- Difficulty in handling immediate values, especially sign extension

Our solution to the first problem was to prohibit buses in datapaths and require students to draw dedicated wires between all components. This tended to make the datapaths much more clear and correct. The second problem was largely solved by explicitly discussing immediate handling in class and showing how it is handled in a variety of existing architectures.

4. Student Response

4.1 Surveys

In addition to the qualitative feedback, quantitative feedback on the project was solicited from the first class of electrical engineering students and the first class of computer science students. The electrical engineering students were asked to write an evaluation of the project as part of their subproject 3 report. Of the 16 project teams, 4 teams called the project “enjoyable”, 6 described it as “a good learning experience”, 2 stated that it “made them better engineers,” and 1 team described it as “the best project since they began college.” 3 teams described the project as “generally recommended”, but felt it needed to give more guidance in instruction set design.

The computer science students were given a survey of 11 questions which they submitted anonymously along with the course evaluation, of which 6 asked for a numerical response. The mean response, on a scale from 1 to 5 (with 5 being most and 1 being least), is shown in Table 1.

How well did the projects help you learn the overall course material?	4.38
How well did the projects help you visualize how processors work?	4.63
Did you enjoy the projects?	4.50
Were the projects appropriate to your background as a computer scientist?	4.43
Do you feel that the projects gave you an opportunity to think creatively?	4.50
Was the division of the overall project into three components helpful for you?	4.13

Table 1. Student evaluations of the project

4.2 Advanced Work

7 of the 8 computer science students and 13 of the 16 teams (35/44 students) in the first class of electrical engineering students completed the project successfully. Of those, 5 computer science students and 6 electrical engineering teams (17 students) completed significantly more than the project required. Examples of advanced work included highly creative instruction sets, assemblers, and a hardware implementation.

5 computer science students and 5 electrical engineering teams experimented with instruction sets vastly different from the MIPS instruction set they had studied before. These instruction sets included stack-based architectures, accumulator-based architectures, and instruction sets with multiple levels of indirection.

5 computer science students and 1 electrical engineering team wrote assemblers for their architecture. The assemblers parsed their assembly language, performed syntactical error checking, and produced a machine code file readable by their simulator. The students demonstrated their assemblers for a variety of programs.

One computer science student successfully designed a three-stage pipeline for his processor in addition to a nonpipelined design. The students evaluated the performance increase from the nonpipelined processor to the pipelined one.

A team of four electrical engineering students implemented their processor in hardware for independent study credits. The students redesigned their machine in Verilog, wrote a softcoded control ROM for their processor, implemented the processor on an FPGA, and interfaced it with LEDs and switches. The students demonstrated their processor for several different programs.

Acknowledgements

The author would like to thank his students for giving permission to publish screenshots of their project work as long as they remained anonymous. The author also thanks the reviewer for the helpful comments and corrections.

References

[1] L. Kalampoukas, A. Varma, D. Stiliadis and Q. Jacobson, "The CPU Design Kit: An Instructional Prototyping Platform for Teaching Processor Design," Workshop on Computer Architecture Education, Int'l Symposium in Computer Architecture, 1995.

[2] T. Stanley and M. Wang, "An emulated computer with assembler for teaching undergraduate computer architecture," Workshop on Computer Architecture Education, Int'l Symposium in Computer Architecture, 2005.

[3] L. Udugama and J. Geeganage, "Students' Experimental Processor: A processor integrated with different types of architectures for educational purposes," Workshop on Computer Architecture Education, Int'l Symposium in Computer Architecture, June 2006.

[4] Course Syllabus for CS316 at Cornell University. http://www.cs.cornell.edu/courses/cs316/2006FA/projects123/

[5] Course Syllabus for CSSE 232 at Rose-Hulman Institute of Technology. http://www.rose-hulman.edu/class/csse/csse232/0506b/www/index.shtml