The Mythical Creature Approach - A Simulation Alternative to Building Computer Architectures

Peter Jamieson[1], Darrel R. Davis[2], and Brooke R. Spangler[3]
Miami University, Oxford, OH, 45056
[1]Dept. of Electrical and Computer Engineering
[2]Dept. of Educational Psychology
[3]Dept. of Psychology
Email: jamiespa, davisde, spanglbr@muohio.edu

Abstract— *In this paper, we present a method to help teach computer architecture (or computer organization) by developing an in class system where the students, themselves, compile high-level code and simulate the execution of their compiled programs. We call this simulation framework* O*ur* R*ealistic* C*omputer* S*imulation (ORCs), and in this paper, we will describe how this framework is developed throughout a semester to help students understand the increasing complexity of a computer architecture. At the end of the course, the ORCS framework can be used by students to test out their own optimizations, and the simulation helps them determine if these optimizations improve the execution of a program on their machines.*

Keywords: Computer Architecture, Simulation, Teaching

1. Introduction

Computer architecture and computer organization courses are taught to first, second, and third year students in almost all curricula in computer science and computer engineering. The pedagogical approach to teach these courses is to provide students a perspective on the history, the organization, the languages, and the operation of processors and the surrounding system. The overall goal is that students will complete such a course with a reasonable understanding of computation on a sequential processor occurs.

There are a variety of approaches on how to provide a student with an understanding of how a processor works. Of late, with the increased capacity and speed of Field-Programmable Gate Arrays (FPGAs) and the low cost point for buying FPGA development kits (between 100 USD and 300 USD), we are beginning to see students building a processor in hardware [1]. This is, in our opinion, the best approach for students to understand the detailed working of a processor and accompanying assembly program, but there are a number of scenarios where either the curriculum or the institution does not allow such an activity. For example, in our Computer Architecture course at Miami University, there is no time set aside in the students schedule for a lab period and the computer science students do not have a background in digital design.

For this reason, we have developed an in class architecture simulation environment in which the students take on the functional roles of a computer and execute various programs. In this paper, we describe the progression of this in class computer simulation, which we call **O**ur **R**ealistic **C**omputer **S**imulation (ORCs). This progressions exposes students to the increasing complexity of a computer architecture, allows students to try optimizations at the compiler, architecture, and memory hierarchy level, and provides students with the opportunity to see how a computer works without having to deal with many of the low-level details required to actually build a real machine.

The remainder of this paper is organized as follows. Section 2 provides a brief description of some approaches to teaching computer architecture in an undergraduate setting. Section 3 describes our approach to teaching computer architecture and lists some of the concepts we hope the student will understand in the course. Section 4 describes the ORCS framework and describes the progression of the framework from a very simple machine to a more complex machine that includes some modern optimizations. Finally, section 5 concludes the paper with a brief discussion of students perception of this approach.

2. Background

Two of the most popular textbooks used to teach undergraduate students computer architecture are Patt and Patel's, "Introduction to Computing Systems - from bits & gates to C & beyond" [2] and Patterson and Hennessy's, "Computer Organization and Design - The Hardware/Software Interface" [3]. Our opinion is that both of these books are excellent accompanying textbooks that expose students to computer architecture.

Patt and Patel's book is a bottom up approach to learning about computer architecture. The textbook starts by describing the digital components that make up the system, and then continues with more and more complex topics in computer architecture until the student begins to understand how a high-level program, written in the C language [4], is mapped to a processor and executed. Their approach is accompanied

with the LC-2 computer simulator and tools that allows students to develop assembly programs and execute them on this machine. The instruction set architecture (ISA) is created for the LC-2 machine with a focus on providing the learner with a comprehensible language, but this ISA is not used in commercial processors.

Patterson and Hennessy's book approaches the same topic by providing a similar bottom approach that can be customized for students depending on if they are hardware or software focused. Their book is accompanied with software for the SPIM simulator [5] which simulates the MIPS [6] ISA. Patterson and Hennessy are also popularly known for a quantitative approach to learning computer architecture in their book, "Computer architecture: a quantitative approach" [7], which is more focused towards students who will actually be working in the computer architecture field.

Many courses using either of these textbooks tend to take the approach of lecturing the course material in class and assigning homework to build various assembly programs that achieve a particular task. For example, programs such as basic computation, function calls, and pointer arithmetic are commonly assigned to expose the students to how high-level languages are converted to assembly programs.

For the most part, this approach has been the preferred method of choice since, other than building a simulation of a processor, technology did not permit the actual building of a processor. Recently, the FPGA, which is a programmable hardware chip, has shown significant improvements in the number of logic gates it contains and the speed at which these chips operate at. Companies, such as Intel, are using FPGAs to prototype, emulate, and test their upcoming products [8]. Additionally, Altera [9] and Xilinx [10] (the top two FPGA companies) now provide university programs accompanied with development kits that can be bought in the range of 100 to 300 USD. These development boards can be used as a platform for students to build a complete computer system. For example, Black, in 2008, published a paper [1] that described step-by-step a set of labs to implement an 8085-based processor.

We believe that the "I built it" approach as opposed to the "I used it" approach to teaching computer architecture is great way for students to learn and be exposed to computer architecture. Unfortunately, there may not be enough time or resources in all computer architecture courses to perform such an exercise, and for this reason we have been developing the ORCS framework as described earlier to expose the students to building and executing programs on a machine. Next, we describe the goals of our computer architecture course and the constraints on the course in which our ORCS framework is used in.

3. Computer Architecture at Miami University - Our Goals and Constraints

The computer architecture course at Miami University (CSA 278/ECE 278) is offered as a 200 level course and is cross-listed between the Computer Science and Software Engineering Department and the Electrical & Computer Engineering Department. Students with majors and minors within either of these two departments are likely to take the course. For this reason, there is a great diversity in the students in computer architecture when it comes to their previous experiences in software and hardware design. The prerequisites for the computer architecture course include introduction to programming and data structures, meaning all the students have been exposed to programming languages (normally JAVA programming).

The goals of the computer architecture are listed here `http://www.eas.muohio.edu/?id=9244` and require that once the students complete the course they have the ability to:

1) describe the operations performed by the CPU.
2) enumerate the registers in a CPU and describe their uses.
3) convert unsigned integers between the following representations: decimal, binary, octal, and hexadecimal.
4) represent signed integers using one's and two's complement representations.
5) perform addition and subtraction of signed integers represented in two's complement representation.
6) describe the salient aspects of the stored program concept.
7) describe commonly used instructions, their formats, operands required, and encoding to opcodes.
8) describe the various memory addressing modes used by the instructions with example usage.
9) describe the concepts and relationships between physical and virtual memory.
10) describe the key components of a CPU and their functionality.
11) explain the concept and use of microprogramming
12) describe the process of interrupt handling.
13) describe the various phases of assembly.
14) illustrate the relationship between mnemonics and machine language translation.
15) describe the passes of an assembler and illustrate the use of various the data structures used internally by the assembler.
16) use selected assembler directives for assembly language programming.
17) develop assembly language programs, debug assembly programs, and trace the operation of assembly language programs.
18) describe the process of linking and loading programs.
19) describe the four stages in a traditional pipeline
20) describe the concept of CPI and quantitatively compare performance of various architectural solutions.
21) describe the taxonomy and categorization of computers into SISD, SIMD, MISD, and MIMD architectures.
22) describe the concept of superscalar processors
23) describe conventional bus and network connections.

This is not the complete list of all the course outcomes, but even this reduced list is a significant number of outcomes. We believe that by using the ORCS framework, so that the student simulates various assembly programs, that all but

items 9, 12, 20, 21, 22, and 23 are thoroughly covered and practiced using this approach. We believe that the items that are not covered thoroughly in our approach, however, are easier to describe and understand by describing them in terms of the ORCS framework. For example, a student who has significant experience in being part of and compiling programs for their ORCS system can be taught the concept of interrupt handling by illustrating how an interrupt is similar to a function call with some additional hardware.

This computer architecture class runs in both the fall and spring terms over a 15 week period. In the spring semester, when we use the ORCS environment for teaching, the class meets three times a week for fifty minutes. We dedicate the third class in each week to exclusively having the students simulate their respective machines, and during the other two classes we occasionally run simulations if there is a particular point that is best demonstrated in this fashion.

4. The ORCS framework

The ORCS framework, which stands for **O**ur **R**ealistic **C**omputer **S**imulation, is named based on a mythical being. We chose this naming since it fits into our belief that for the machines in our world that we don't understand how they work, we tend to believe that the machine works by magic or some small gnome, elf, or orc is inside the machine and performs our requested actions. A computer architecture, in our case, is the machine of interest, and as we progress through the course adding increasing complexity to our computer architecture, the mythical creatures that perform various actions within the processor are removed. In many ways, these mythical creatures performing specific actions within a machine is in a way an abstraction level.

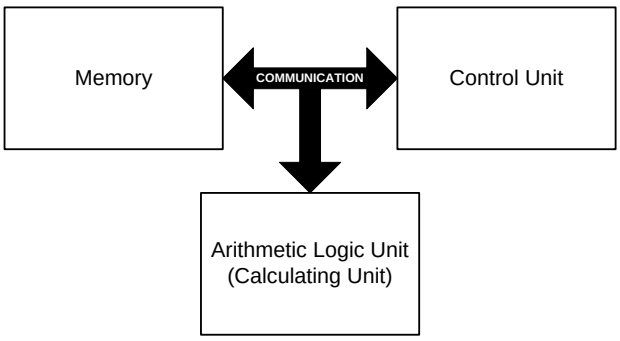

Fig. 1
SHOWS THE BASIC VON NEUMANN ARCHITECTURE WITHOUT THE ACCUMULATOR

Figure 1 shows the first structure of a processor that we start with for the ORCS framework. This is similar to the basic architecture proposed by Von Neumann [11]. Basically, we take the approach where data and the program are both stored in the memory, and a controller moves information around the system based on an interpretation of the program instructions. The students will be acting as four parts of this machine - one each for the controller, calculator, memory, and messenger. Therefore, we divide the class into groups of 4 students, and each group is responsible for compiling and executing a program.

Figure 2 shows more details of the starting architecture for the ORCS framework. In this figure, we see that the controller shows has program counter (PC) and a four step process for executing each step of the program. The memory is introduced as a simple table that has an address on the left side and associated data contained in the right cell. We have loaded a sample C program into memory that we use as our first program to illustrate to the students the execution steps of a simple processor. Note that the "printf" function call, in this program, is introduced to the students as a much more complex library function and is simply illustrated as a program counter jump. There are no details provided for the calculation unit, and it is assumed that this unit can perform any arithmetic operation at this time.

At this development stage, C programs are introduced to the students that declare variables, include pointers, execute loops, execute conditionals, and perform calculation on integers and strings. The students are then responsible for compiling each program into memory and then executing the program in simulation mode where each of the four members take on a role of the machine. During execution, nobody is allowed to talk, and the students uses pieces of paper to pass messages to each unit (via the messenger) and can record information down on pieces of paper. To start out, there is no assembler or machine language used, and instead, the programs are stored as C statements.

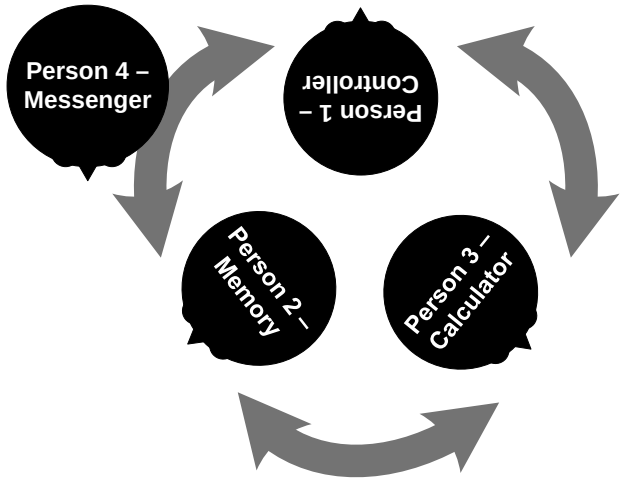

Fig. 3
BIRDS EYE VIEW OF THE STUDENT POSITIONING FOR THE BASIC ORCS MACHINE

Figure 3 shows how to physically place the students for

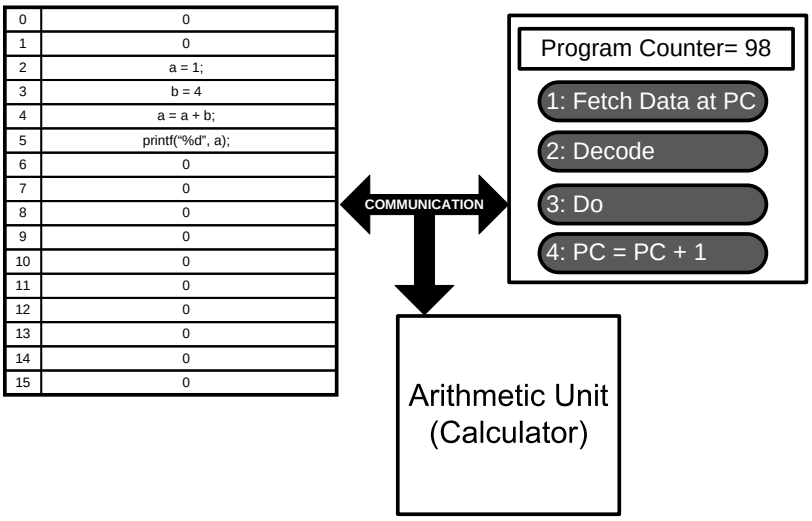

Fig. 2
MORE DETAILS FOR THE FIRST MACHINE

simulating their machines. The three units are made to sit facing away from one another, and the messenger person goes around the circle delivering messages. The reason that such a setup is needed is to discourage non-verbal communication, which may occur when students become aware that their programs and actions have caused an error to happen. However, for the first few sessions when introducing the simulation idea to the class, we allow the students to see each other and talk while they execute a program. This allows the group to discuss what is happening and to get an understanding of how the machine executes a program.

The base ORCS framework is then modified as the course progresses to include more detailed simulation aspects of an architecture as various topics are introduced. For example, the first topic in the course after introducing the basic architecture is number systems and addition/subtraction in 2's compliment. Once this topic has been introduced, all the addresses in the memory are now labeled with a hexadecimal number in the form 0x00af, and all operations in the calculator unit need to be done in terms of binary addition/subtraction. Next, we introduce registers and memories in the course, and this results in introducing bit-widths to the memory and adding a register file to the ORCS framework. The students are free to choose where they would like to add registers to their machine.

Figure 4 shows how the architecture evolves to include a memory using binary and hexadecimal numbers and the inclusion of registers. These updates to the simulation model are maintained throughout the term, and the students gain significant experience dealing with these number systems. The reason for this is that they will constantly have to use number systems, including two's complement number representation, to perform any calculations and memory stores. After a few weeks, these number system conversions become second nature.

The ISA can be introduced to the ORCS framework at any point and can either be a defined ISA (such as MIPS) or can be the instructors/students invention. Our approach is to first allow the students to define a basic instruction set that includes memory loads and stores, register movement, a few calculator instructions, a jump command, and a conditional jump. We also introduce the MIPS ISA in the sixth week of the course and at the same time ask the students to use this ISA in their machines. This transition point is a good point at which the instructor can discuss the historical debate between complex instruction set computers (CISC) and reduced instruction set computers (RISC). Later in the course (in the last 4 weeks) when machine optimizations are introduced, we allow the students to use either ISA. This results in various groups using different ISAs, and as a class we can debate over why different ORCS machines are executing programs faster than one another.

The complexity of the ORCS framework is increased for approximately two thirds of the course. For the remaining one third of the course, we introduce optimization concepts such as pipelining, instruction level parallelism, and memory hierarchy. At this point, we allow the students to try and optimize their own ORCS machine so that it can execute programs faster compared to others approaches. To achieve some optimizations, such as pipelining, the merging of groups is allowed so that groups provide enough execution units to simulate/execute the computation. For example, a three stage pipeline will require one additional student to implement the write-back stage (and may require an

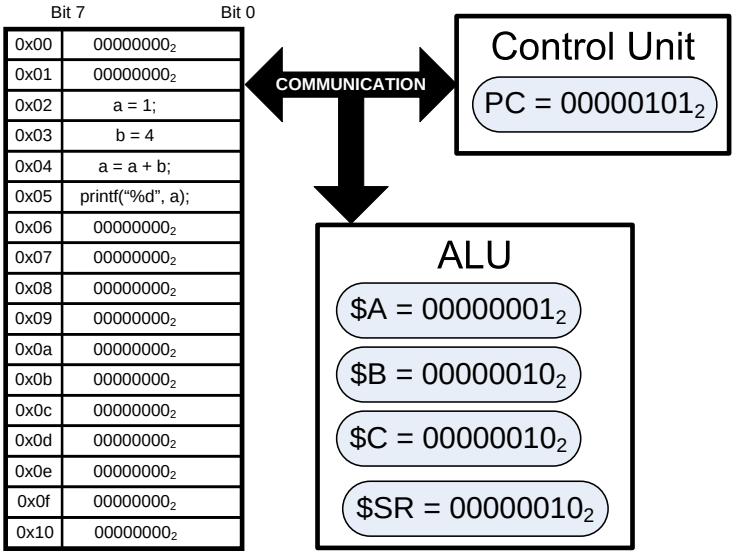

Fig. 4
AN ORCS MACHINE WITH MEMORY DETAILS AND REGISTERS

additional student to implement a simultaneous read and write memory).

Our time line for evolving the students machines in the ORCS framework over the 15 weeks is the following:

- Week 1 - Base machine introduction (Figure 1)
- Week 2 - Base machine with pointers, loops, and conditionals
- Week 3 - Memory, registers, and binary add and subtract (Figure 4)
- Week 4 - Assembler instructions and stricter communication protocol
- Week 5 - Exam week
- Week 6 - Competition to see who's machine is the fastest and why
- Week 7 - Introduce MIPS ISA
- Week 8 - Addressing modes in MIPS
- Week 9 - Introduce a cache structure and experiment (discussed below)
- Week 10 - Exam week
- Week 11 - Clock speed experiment (discussed below)
- Week 12 - Compiler optimizations
- Week 13 - Pipeline machines
- Week 14 - Any optimization allowed including parallel machines - who can make the fastest machine
- Week 15 - Review week

This is one way of evolving computer architecture within the ORCs framework, but there is plenty of flexibility in changing this presentation order.

4.1 Some Interesting Experiments

There are a number of experiments that can both illustrate a point and are fun for the students.

4.1.1 Clock Speed and Time per Instruction

One experiment to try once the students have become familiar with the ORCS system is to try a clock speed experiment. Essentially, the students are assigned anywhere from 1 to N benchmarks, and are given a week to compile their program and practice executing it. These benchmarks should be created in a way such that depending on the data different paths of execution will be taken.

On the experiment day, the instructor brings a stop watch to class, a metronome if you have one, and a data set that the students have not seen yet. Then have each group execute one of the benchmarks while being timed. Ensure that the final executed results are correct, and then allow the students to calculate on average how long each instruction took their machine to execute. At this point you can ask questions about the speed of their machine and where are the bottlenecks in the system.

Next, ask the students to break up their machine into a set of defined actions. Finally, ask the students to tell you what they think is the slowest task in their machine and how long this task will take. If you can, set the metronome at the slowest speed of the machine and ask them to simulate a program at their so called "clock speed" seeing if they can maintain the required cadence. Alternatively for a larger class, set the metronome at a certain speed and see if everyone in the class can execute at that rate. Increase the rate of the metronome until all the machines fail. This approach is modeled on the beep test used to determine a set of athletes VO2 max (`http://en.wikipedia.org/wiki/Multi-stage_fitness_test`).

4.1.2 Cache Replacement Policies

If your course introduces memory hierarchy and caches, then the ORCS framework has the potential for students to

Table 1
RESULTS FROM THE STUDENT SURVEY ABOUT THE ORCS FRAMEWORK

Question	Mean	Standard Deviation
1. ORCS helps me understand how C programs compile to Assembly	2.43	1.07
2. ORCS helps me understand how a processor executes a program	2.32	1.17
3. ORCS helps me understand all the main pieces of a computer architecture	2.57	0.97
4. I would rather not have use ORCS and replace them with more lecture time	2.48	1.14
5. I would rather write more assembly programs	2.23	1.21

learn about cache replacement policies. Again, develop 1, 2 or 3 benchmark programs that touch the memory in various patterns. Briefly describe the concepts of cache replacement policies. Allow the students to work for a few days on implementing and researching various cache replacement policies.

Within the ORCS framework include a student to simulate the memory and a student to simulate a cache with approximately four cache spots. On the experiment day, introduce some other benchmark programs with similar and different memory access patterns. Have the students execute these programs using their best cache replacement policy and have them calculate cache misses and cache hits.

4.2 Student's Experiences

To get a feel for the student's perspective of this approach to teaching architecture we surveyed the students in our computer architecture class with IRB approval. We understand that the student's responses to a survey is more of a general feeling towards the approach as opposed to an comparative opinion between different methods of learning computer architecture, and therefore, the scientific value of these survey results is very low. However, it is interesting to see how the students respond to an activity that is very foreign to their learning experience.

For this survey, the students were given the option at a midterm evaluation of the course to answer 5 additional questions related to the ORCS aspect of the course on a scale of 0 to 4 where 4 is strongly agree and 0 is strongly disagree. At this point the students have participated in 6 ORCS simulation sessions. The evaluations are anonymous and have no impact on the students assessment in the course.

Table 1 shows the results of this survey and the 5 questions that were asked. A total 12 students responded to this part of the survey out of a total of 36 students (one third participation). The survey results show that the students don't have a clear perspective whether the ORCS simulation activities impact their learning of the material. These results are not unexpected as we don't believe the students have attempted a significant amount of meta thinking as to how this approach impacts their learning.

5. Conclusion

In this work, we have introduced our ORCS framework that allows students in a computer architecture course to experience building a processor and execute programs by the students themselves simulating various programs. This approach provides the students with significant exposure to, practice of, and experimentation with the many concepts and ideas introduced in an undergraduate computer architecture course.

We surveyed traditional approaches to teaching computer engineering pointing out that it is now possible for students to build a processor on an FPGA. The ORCS framework is an in class alternative to a processor building approach with the benefit that the students do not have to deal with many of the low-level details needed to be learned and understood to build such a system. We do not claim that our method is better than any other existing pedagogical approach, but offers teachers an alternative to expose students to computer architecture.

References

[1] M. Black, "Building a computer from scratch: a hardware lab sequence for computer science students," *J. Comput. Small Coll.*, vol. 24, no. 3, pp. 32–38, 2009.

[2] Y. N. Patt and S. J. Patel, *Introduction to Computing Systems: From Bits & Gates to C & Beyond*. New York, NY, USA: McGraw-Hill, Inc., 2004.

[3] D. Patterson and J. Hennessy, *Computer Organization and Design: The Hardware/software Interface*. Morgan Kaufmann, 2005.

[4] B. W. Kernighan and D. M. Ritchie, *The C programming language*. Prentice-Hall, 1978.

[5] J. Larus, "SPIM S20: A MIPS R2000 Simulatora," Tech. Rep., 1990.

[6] J. Hennessy, N. Jouppi, S. Przybylski, C. Rowen, T. Gross, F. Baskett, and J. Gill, "Mips: A microprocessor architecture," in *MICRO 15: Proceedings of the 15th annual workshop on Microprogramming*. Piscataway, NJ, USA: IEEE Press, 1982, pp. 17–22.

[7] J. L. Hennessy and D. A. Patterson, *Computer architecture (2nd ed.): a quantitative approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996.

[8] P. H. Wang, J. D. Collins, C. T. Weaver, B. Kuttanna, S. Salamian, G. N. Chinya, E. Schuchman, O. Schilling, T. Doil, S. Steibl, and H. Wang, "Intel®atom™ processor core made fpga-synthesizable," in *FPGA '09: Proceeding of the ACM/SIGDA international symposium on Field programmable gate arrays*, 2009, pp. 209–218.

[9] "Altera University Program at http://www.altera.com/education/univ/unv-index.html," 2010.

[10] "Xilinx University Program at http://www.xilinx.com/univ/," 2010.

[11] J. v. Neumann, "First draft of a report on the edvac," Tech. Rep., 1945.