VN-Sim: A Way to Keep Core Concepts in a Crowded Computing Curriculum

R. Raymond LANG
Computer Science, Xavier University of Louisiana
New Orleans, LA 70125 USA

and

Theresa BEAUBOUEF
Computer Science, Southeastern Louisiana University
Hammond, LA 70402 USA

ABSTRACT

Contemporary computer science curricula must accommodate a broad array of developments important to the field. Tough choices have to be made between introducing newer topics and retaining fundamentals that ground the discipline as a whole. All too frequently, understanding of low level coding and its relation to basic hardware is sacrificed to make room for newer material. VN-Sim, a von Neumann machine simulator, provides a mechanism for streamlined coverage of low level coding and hardware topics.

Keywords: von Neumann machine, simulator, machine language, assembly language, low level programming

1. INTRODUCTION

The field of computer science has come a long way from the programming of large computers with limited memory and instruction sets to the smaller but more powerful computers of today. Likewise, programming applications also have changed to reflect this continuing impact of Moore's Law [4, 5], allowing for applications in a wide variety of fields.

Computer science and programming have evolved through the years, moving more from a mathematical perspective to a development perspective, where programmers have a wide array of tools available. In practice, developers adapt existing packages to solve problems. Mathematics and problem solving skills continue to be important [1, 2], yet because of the diverse nature of the field of computer science, some technology-related careers paths do not rely as heavily on core computing concepts.

Computer science education has also changed through the years to reflect the needs of business and industry to produce graduates who can integrate packages and put together solutions by using already existing software. They use languages with object libraries built in, so they become proficient in software development environments and in searching for solutions online. Students do learn how to code, they learn about hardware, and they learn about systems. However, in this age of abstraction and code reuse, students often do not gain a fundamental understanding of the very basics of computer science and how hardware and software come together at a low level to perform simple calculations. This is as important to the computer scientist as the knowledge of atomic structure is to the chemist or cell functionality is to the biologist.

With the VNSim package, students can interactively see how code that they write is stored and implemented in hardware. They can view memory contents, and they can learn about errors that can occur in low level coding, which eventually can cause errors in high level programming applications. We give examples of the VN-Sim, and describe how it can reinforce computer science skills for beginning and more advanced students.

2. UNDERSTANDING BASIC HARDWARE AND MEMORY

Instruction in hardware fundamentals begins with the von Neumann model: a central processing unit (CPU) and a memory storing both programs and data. Student understanding of the relationship between the control unit (CU) and the arithmetic-logic unit (ALU) is important preparation for concepts such as data representation, control flow, indexing, digital logic, and more.

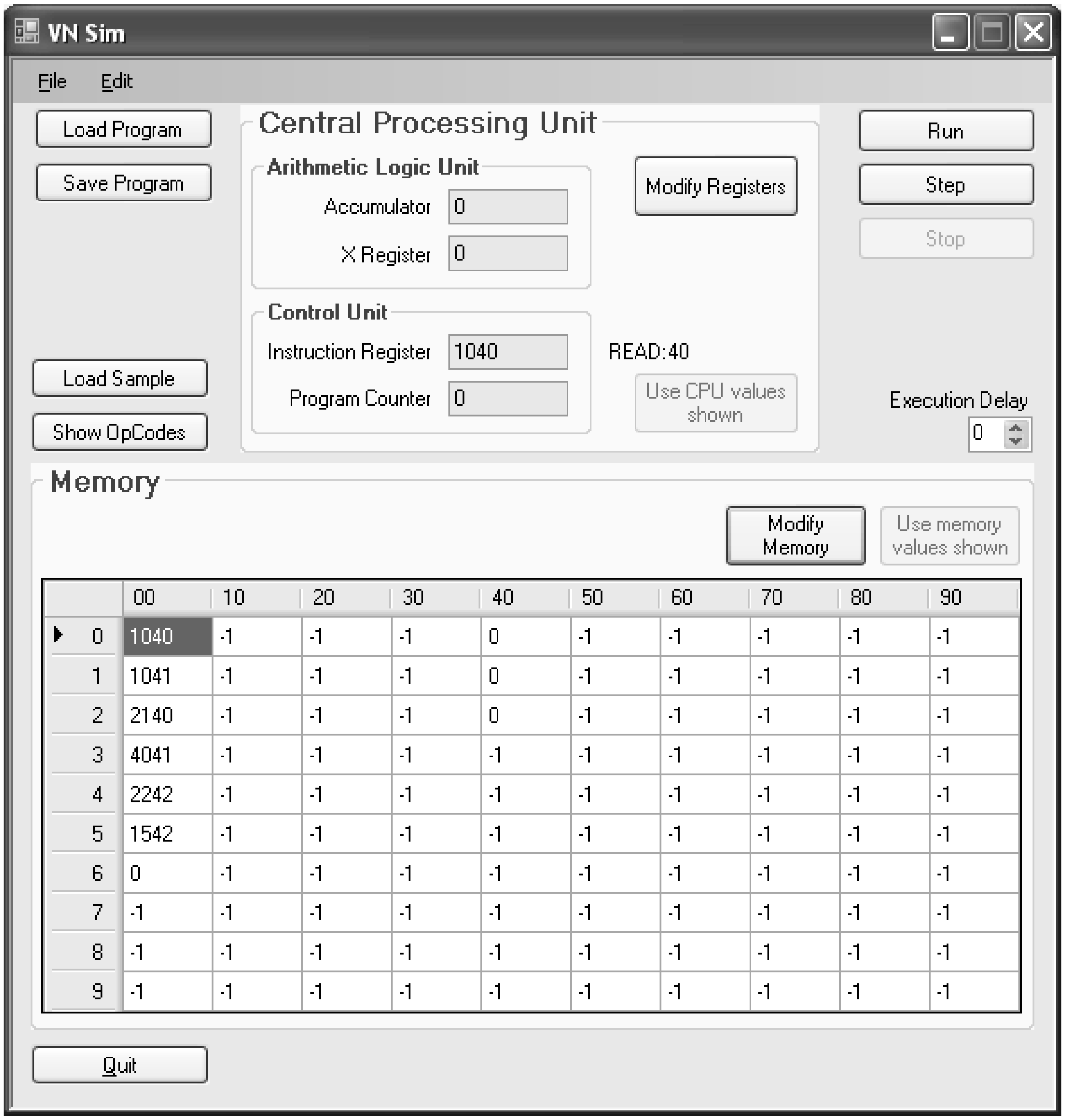
A static description of the von Neumann architecture [3] consists of the ubiquitous box and arrow diagram showing the connections among the components. The fetch-decode-execute cycle conveys how a stored-program computer operates. To go beyond the five-minute hand-wave of this topic, an instructor must describe a few instruction codes, arrange them in memory, and perform a hand execution of a program of up to a dozen lines or so. But such a presentation risks leaving students with the impression that a von Neumann machine is too simple to do anything but just the sort of programs that can be hand-traced in just a few minutes. VN-Sim is a von Neumann simulator that can provide convincing demonstrations of the power and scope of the model. It supports a better understanding of stored-program computers by allowing direct manipulation and observation of a working example of the von Neumann architecture.

The execution of VN-Sim is governed by its instruction set. There are instructions for clearing, loading, and storing the ALU registers, for addition and subtraction in the accumulator, and for incrementing and decrementing the x-register. The branching instructions are an unconditional jump, two jumps conditioned on the accumulator, and one jump conditioned on the x-register. The READ instruction stores into memory a value provided by the user, and the WRITE instruction displays the contents of memory to the user. HALT instructs VN-Sim to do just that. The instruction set and the I/O dialogs are shown in figures 2, 3, and 4.

The opcodes are four digit decimal codes in which the leftmost digits signify the operation, and the rightmost digits the operand, a two-digit address in the VN-Sim's memory. Only some instructions require an operand. In Figure 2, the operation codes are shown, followed by two dashes for operations without an operand or two plusses for operations with an operand. VN-Sim performs absolute addressing only.

3. UNDERSTANDING LOW LEVEL PROGRAMMING

A solid understanding of high-level programming languages is grounded in a grasp of operations at the lowest levels. VN-Sim illustrates how machine level operations can be arranged to perform a variety of tasks.



Accumulator: 0

X Register: 0

Instruction Register: 1040

Program Counter: 0

READ:40

Execution Delay: 0

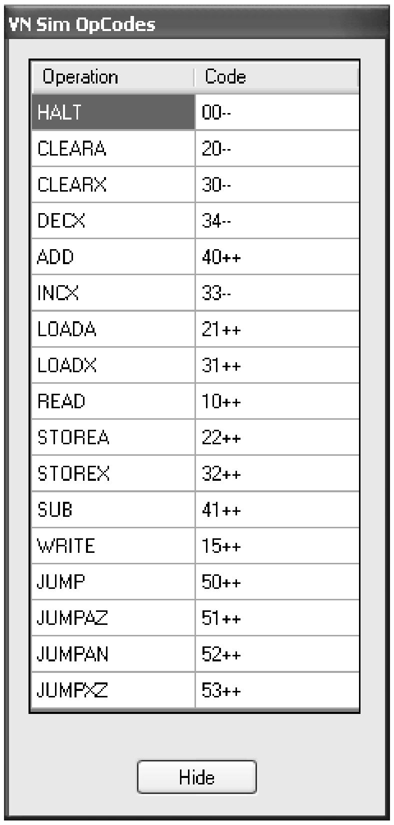
	00	10	20	30	40	50	60	70	80	90
0	1040	-1	-1	-1	0	-1	-1	-1	-1	-1
1	1041	-1	-1	-1	0	-1	-1	-1	-1	-1
2	2140	-1	-1	-1	0	-1	-1	-1	-1	-1
3	4041	-1	-1	-1	-1	-1	-1	-1	-1	-1
4	2242	-1	-1	-1	-1	-1	-1	-1	-1	-1
5	1542	-1	-1	-1	-1	-1	-1	-1	-1	-1
6	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
7	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
8	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
9	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Figure 1: VN-Sim main window. Memory addresses 0 through 5 contain a program to add two numbers entered by the user and display the sum.

The simulator contains a built-in program to add two values; it loads into the first six memory locations when the "Load Sample" button is clicked (see Figure 1). To reinforce the role of the program counter (and lay the groundwork for services provided by an operating system), the user must manually set the value of the PC to the address of a program's first instruction. The PC is initially set to 0 on startup, but to run the sample program more than once, the user must set it back to 0 manually.

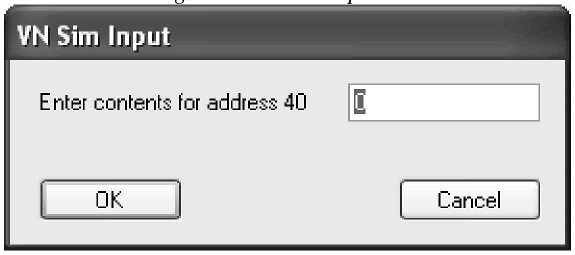
In presenting to students, the symbolic instructions are initially described as a convenience to the human programmer, who must still translate the symbolic instructions by hand to the corresponding numeric opcodes and then enter these directly into the VN-Sim's memory. At this level, the student must decide what memory locations will be used for data storage and take care to use the correct operands when translating the program.

To gain familiarity with the instruction set, students are asked to modify the sample program in a variety of ways: by changing the storage locations used by the program, by making the program subtract instead of add, and so forth. Students are introduced to the jump instructions by means of a short program to read two values and output the larger. After gaining some appreciation for directly manipulating memory, students begin using files on the host system to save or load the contents of a range of memory.


VN Sim OpCodes

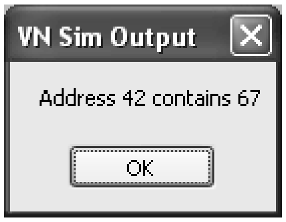
Operation	Code
HALT	00--
CLEARA	20--
CLEARX	30--
DECX	34--
ADD	40++
INCX	33--
LOADA	21++
LOADX	31++
READ	10++
STOREA	22++
STOREX	32++
SUB	41++
WRITE	15++
JUMP	50++
JUMPAZ	51++
JUMPAN	52++
JUMPXZ	53++

Figure 2: VN-Sim Opcodes.


VN Sim Input

Enter contents for address 40

Figure 3: VN-Sim input.


VN Sim Output

Address 42 contains 67

Figure 4: VN-Sim output.

4. PROGRAM VERIFICATION FOR LEARNING PROGRAMMING

VN-Sim can also be used to help beginning students understand the concepts of programming through the use of code verification. Here students are given some code and the documented requirements for the code and asked to verify that the code does what it is intended to do. Simpler programs are given at first in order to familiarize the students with basic coding concepts. However, more complex code is given shortly after, with the goal of exposing the students to high-level coding constructs, proper documentation and coding techniques, and problem solving. Through the verification of existing code, beginning students can rapidly learn the basics of coding and programming style.

5. CONCEPTS FOR MORE ADVANCED STUDENTS

Once students have mastered the basic skills of the VN-Sim such as the use of registers, addressing, comparisons, and jumps, they can begin to expand on these concepts to learn about such things as multi-path selection or case statements, looping constructs, function calls, and memory management. Many of these concepts can be illustrated in the VN-Sim, even with its limited memory, registers, and instruction set.

Other tasks require the use of additional registers for maintaining a stack base address and stack pointer, for example. Students can attempt to solve certain problems that will lead them to identify additional system resources necessary for their implementation. Students will also gain an understanding of techniques for working with limited resources which will give them insight into problems encountered in the programming of real-time and embedded systems. This should also lead to an appreciation for the abundance of system resources available to programmers and systems engineers today.

6. INITIAL RESULTS

VN-Sim has been used thus far in two courses: (1) a breadth-first introduction to the computing discipline which students take prior to their first programming course, and (2) a senior level programming languages course. In the introductory course, VN-Sim was used to illustrate the following concepts:

- von Neumann architecture
  - the central processing unit, including the roles of the CU and of the ALU
  - a random access memory storing both programs and data
- the fetch-decode-execute cycle
- flow of control, esp. the use of sequence, selection, and repetition in programs
- machine language, assembly language, and the distinction between the two
- input and output mechanisms
- the distinction between operations and operands
- low level programming of small arithmetic operations, esp. performing operations that are not provided in the instruction set, e.g. multiplication
- debugging and code tracing

Assessment was done by in-class exercises, out of class homeworks, and quiz questions. Students were asked to describe the von Neumann architecture, define key concepts, predict the output of short programs, and to translate from assembly code to machine code and vice versa. Three weeks of class time was spent on this material.

Students found the material moderately challenging, and the grades bore this out. This was our first attempt presenting this material in this context, so there was no previous data to compare learning results.

In the programming languages course, students were given an assignment to write a symbolic assembler targeting the VN-Sim instruction set. About two thirds of the enrolled students successfully completed the assembler or had only minor flaws, the remaining had major shortcomings or were not submitted. VN-Sim supports only absolute addressing, and the students were able to grasp the importance of relative addressing and gain greater understanding of the flexibility provided by relocatable code modules.

7. FUTURE WORK

Plans are underway to use of VN-Sim in the computer organization course. This course combines a previous assembly language course that some felt was obsolete with additional concepts necessary for computer engineering technology students. VN-Sim is ideal for such a course as it illustrates assembly language program concepts, basic computer hardware and architecture components. We expect students in this course will be able to better visualize and learn about low-level workings of the computer and to program basic tasks. Because many students in this course are engineering technology majors, they are not as versed in programming as their classmates who are computer science majors. These students should benefit from the hands-on, virtual machine approach used by VN-Sim.

8. CONCLUSION

Computer science brings together many areas of science, technology, communication, and human relations. Computer applications today are powerful and sophisticated, incorporating graphical user interfaces and a variety of hardware devices and networking techniques. When all is said and done, however, a computer is still a simple machine. It can store data, it can add, and it can compare two values. Every other operation is built off of these basic abilities, so an understanding of the low-level concepts related to basic computer hardware and programming is essential for computer scientists.

In this paper we described the VN-Sim system and how it can be used to enhance the education of computer science students. It reinforces the fundamentals of hardware and software and their interrelationship in what we call programming. Students benefit from the hands-on approach as they examine results of instructions and learn how they can program their own instructions to achieve desired results. The VN-Sim system has proven to be both easy to learn and an effective teaching tool.

ACKNOWLEDGMENT

This research was sponsored in part by the NSF grant CCF-0939108.

9. REFERENCES

[1] Beaubouef, T., Why Computer Science Students Need Math**, SIGCSE Bulletin (inroads)**, 34, (4), , 57-59, 2002.

[2] Beaubouef, T., Lucas, R., Howatt, J., The Unlock System: Enhancing Problem Solving Skills in CS1 Students**, SIGCSE Bulletin (inroads)**, 33, (2), 43-46, 2001.

[3] Godfrey, M., Hendry, D., "The Computer as Von Neumann Planned It," **IEEE Annals of the History of Computing**, Vol. 15, No. 1, 1993, pp. 11-21.

[4] Moore, G., "Cramming More Components Onto Integrated Circuits," **Electronics,** vol. 38, No. 8, 1965.

[5] Stokes, Jon, "Understanding Moore's Law," (Feb. 20, 2003), Retrieved Nov. 15, 2010,

[6] http://arstechnica.com/hardware/news/2008/09/moore.ars/1