AC 2011-774: ACTIVE LEARNING EXERCISES IN COMPUTER ORGANIZATION AND ARCHITECTURE

Jeffrey A. Jalkio, University of Saint Thomas

Jeff Jalkio received his Ph.D in Electrical Engineering from the University of Minnesota and worked for thirteen years in industry in the fields of optical sensor design and process control. In 1984, he co-founded CyberOptics Corporation, where he led engineering efforts as Vice President of Research. In 1997 he returned to academia, joining the engineering faculty of the University of St. Thomas where he teaches courses in digital electronics, computing, electromagnetic fields, controls, and design.

Dan R Schupp

Dan Schupp is finishing his bachelors in Electrical Engineering and Physics at the University of St. Thomas. He has worked with students as a TA for nearly two years, covering topics ranging from introductory engineering courses to design with microprocessors. He is currently employed at Xollai LLC, a robotic vision company.

Active Learning Exercises in Computer Organization and Architecture

Abstract

Current computer science and computer engineering students have grown up using computers on a regular basis. However, they often enter college with no knowledge of how a computer functions and frequently with substantial misconceptions regarding their functioning. The earlier these misconceptions can be replaced by a more accurate model of the computer's operation, the more readily the student will be able to integrate computer science concepts into their working knowledge of the world.

Research in education has long shown that active learning techniques are particularly effective in helping students to overcome pre-existing misconceptions. In this paper we present an activity for exploring basic concepts of computer architecture and organization. In this activity, students play the role of various computer components such as program counter, instruction register, and act out the process of fetching, decoding, and executing instructions. Through this game-like activity, students are also introduced to the idea of constructing algorithms from simple instructions. Because this activity does not assume prior knowledge of computing or electrical engineering it can be used with a wide variety of audiences. It has been used successfully with engineering, education and liberal arts majors, as well as high school students who have expressed an interest in computer engineering.

Introduction

In January 2008, one of the authors co-taught a seminar on artificial intelligence for University honors students. After the first class session, several of the non-technical students (mostly philosophy majors) expressed concern that they did not understand how computers worked and that therefore they might be at a disadvantage compared to the engineering and computer science students in the class. This led to three questions. First, what level of understanding of computer operation was needed for the class and what level of detail was needed to give these students an appropriate level of confidence? Second, what key concepts from computer science were needed for the class that these students might be missing? Finally, how could these students obtain an appropriate level of understanding of computer operation and of the necessary computer science concepts in a very short period of time (since a great deal of material was already packed into the short seminar). These questions led to the initial development of the simulation presented in this paper, which has since found application with high school students, education students specializing in STEM (science, technology engineering and math) education, and engineering students enrolled in a computer architecture class.

In answer to our first question, it quickly became apparent that while no technical understanding of computer hardware operation was needed for the class, most of our students had used computers nearly their whole life without knowing how they worked. Interestingly, this was true for both technical and non-technical students. The big difference between the two groups was not that the engineering and computer science students understood computer function (most of

them had not yet taken a computer architecture course), but rather that the liberal arts students were troubled by their ignorance while the technical students were not. Clearly, it would be useful to provide the entire class with a basic understanding of how stored program computers function at a very simple level.

In addition, for the purpose of the particular class, we determined that it would be useful to introduce a few basic computer science concepts:

1. A finite set of operations can each be numbered and hence be uniquely identified by bit patterns.
2. One can often solve complex problems using algorithms consisting of a sequence of very simple primitive operations.
3. Measurable quantities can be represented as numbers (and hence bit patterns).
4. Computers of a sufficient level of complexity can simulate any other computer.
5. The speed with which a computer executes its instructions can have a profound effect on an observer's perception of the machine's capability.

Having answered the first two of our three questions, we sought a pedagogical approach to accomplish these goals quickly. Given the assessment data showing the efficacy of active learning techniques[1] we decided to take an active learning approach. Most of these approaches (e.g., problem based learning) would require more time than was available in the seminar, so we decided to use a game to teach the required concepts. Lepper and Malone[2] found that games in the form of computer simulations were very effective in achieving educational objectives. While computer simulations have eclipsed in-class group game playing in most game based pedagogies, we chose to use role-playing to simulate a computer. To accomplish these goals, the small class (15 students) simulated the operation of a simple computer, with one student taking the role of the program counter, another the instruction decoder, a few more as registers, and the rest as memory locations.

Implementation

Figure 1 shows the instruction set of our simulated processor. Note that the decimal instruction encoding has been designed to make decoding easy for students. This is the pedagogical equivalent of orthogonal instruction encoding. The most significant digit of the instruction is an opcode so very few unique operations are possible. These instructions were chosen specifically to make the example program relatively simple while maintaining the ability to point out the key concepts mentioned above. We provide conditional branches, moves between the accumulator and

Instruction Decoding Key (Memory contents : X Y Z) (e.g. 426)

X	Interpretation
1	Copy Y into register A
2	Copy register A to Y
3	Add the contents of register Y to register A
4	Place Z in register Y
5	Subtract Z from register Y
6	if Y=0 place Z in program counter
7	stop !

Y	Interpretation
1	Register A
2	Register B
3	Register C
4	Input
5	Output
6	Program Counter

(so 426 means "Place 6 into register B)

Figure 2 - Instruction Set

registers, addition and subtraction operations, and a halt operation. The wording of the descriptions of each operation turned out to be very important. Copy is used instead of move because students often assumed that move implied placing the contents of the source in the destination and removing them from the source, while copy had the correct connotation. Separate add and subtract instructions were used to avoid the need of introducing negative numbers and their representation. The second digit always specifies a register which might function as a source or destination depending on the opcode. One could also view this as a data memory address and consider our simulated machine to have a Harvard architecture with memory mapped IO. The least significant digit always provides a numeric constant which is used either as a branch destination or a constant to be loaded into a register. It should be noted that this severely limits the range of possible branch destinations. In addition to providing us with a simple instruction set that can be easily understood by students, this encoding helps us demonstrate concept # 1 above, that a finite set of instructions can be assigned numbers and encoded as bits stored in memory. Figure 2 shows the instructions given to the student playing the program counter.

Program Counter Rules

1. Start with N=0

2. Each time the instructor says “clock”, ask the class for the contents of address N, then increment N.

3. Replace N with a different value only when instructed to do so by the Instruction Decoder.

Figure 3 - PC Instructions

Figure 3 shows a program written for our simulated computer. This program occupies only 11 memory locations and simply reads two operands from an input register, multiplies them via repeated addition and writes the product to an output register. The execution of this program helps convey concept #2 above by performing multiplication on a computer that does not possess a multiply instruction. This is a great surprise to non-technical students.

Address	Instruction	Opcode
0	mov in, a	140
1	mov a, b	220
2	mov in, a	140
3	mov a, c	230
4	mov 0, a	410
5	if C=0, goto 9	639
6	b+a ->a	320
7	c-1->c	531
8	goto 5	465
9	mov a, out	250
10	stop	700

Figure 4 - Program Listing

To play the game, first students are selected as instruction decoder and program counter. While this can be done at random, it is useful to choose students who are known to be able to follow instructions. The remaining students play the roles of registers and memory locations. Students playing memory locations are given slips of paper with their address, numerical contents, and the meaning of the instruction. Registers are given slips with the name of their register and

possibly scratch paper and pencils. The instructor projects the instruction decoding rules for the class and plays the role of the clock.

For each instruction cycle, the instructor calls out "clock" and the program counter calls out the address of the next instruction (e.g., "0"). When called upon a memory location calls out their contents (e.g., "140"). Next the instruction decoder decodes the instruction, with the rest of the class checking their work (e.g., "Copy contents of input register to register A"). At this point the registers carry out the indicated operation (in this particular case, the instructor acts as the external IO device, providing the input value of 5). Having completed a cycle, the instructor can issue another clock and the program counter will call out the next address.

Choice of the second input value determines the number of iterations through the loop. Choosing input values of 5 and 3 results in a relatively short program run, yet enough iterations of the loop to accomplish the learning objectives. Students typically first fear that they will loop forever and later realize that the loop terminates. Many students do not realize what the program is accomplishing until the last iteration of the loop.

Once the simulation has completed, the students are asked to calculate the approximate clock frequency based on the number of instructions executed and the time taken. We find that the simulation typically runs at about 0.02 Hz. This leads to a discussion of concept #5 – the effect of speed on the perceived capability of the machine. In turn, this provides an opportunity to discuss the ability of such a machine to manipulate images and other data (concept #3) and to introduce the Church-Turing thesis at an elementary level (concept #4 above).

Results

This game has now passed through several iterations resulting in a game which has been used with several audiences. In addition to being used twice for the original audience of students taking an honors seminar, this game has been used several times with high school students taking an enrichment program on engineering, electrical engineering students taking an introductory computer architecture course, and education students taking a course on engineering and technology for high school students.

Uniformly, the high school and honors seminar students reported that the game was very useful in helping them to understand how a computer works; however, when a qualified instruction decoder could not be easily identified, an alternative is to have each memory location decode their own instruction out loud to the class. Our experience with engineering students has also been positive but for class sizes greater than 18, the class should be divided to ensure that each student plays a role in the game.

We have observed an interesting reaction from engineering students when we use this exercise in class. The students are initially frustrated that they are not given and cannot see the answer immediately. After a few instructions are executed and the students begin to see what is happening, this frustration disappears, replaced by excitement as they figure out what is happening.

The professor teaching the education students found that it was advisable to run the simulation twice with different input values. On the first pass, the students focus on learning the concepts being presented and on the second pass they focus on how they could incorporate the game into their own syllabi. The education students found the simulation confusing during the first pass, but were more comfortable with it on the second.

Based on the results so far, we plan to continue to use this game in course for both majors and non-majors. It is an activity that students enjoy and that achieves its educational goals.

Acknowledgements

The authors would like to thank Dr. Ann Marie Polsenberg Thomas for her willingness to use this exercise in her Fundamentals of Engineering course for education students and for her helpful feedback based on her experience teaching that course.

Bibliography

[1] Stewart, J. and Hafner, R. (1991). Extending the concept of problem in problem-solving research. *Science Education* 75,105-120

[2] Lepper, M.R. and Malone, T.W. (1985). Intrinsic motivation and instructional effectiveness in computer based education. In R.E. Snow & M. J. Farr (eds), *Aptitude, learning, and instruction III, Conative and affective process analyses*. Hillsdale, NJ: Erlbaum.