Combining Learning Strategies and Tools in a First Course in Computer Architecture

Patricia J. Teller, Manuel Nieto, and Steve Roach
The University of Texas at El Paso
Department of Computer Science
{pteller, manueln, sroach@cs.utep.edu}

Abstract When instructors learn about participatory learning strategies such as cooperative, active, problem-based, or team-based learning, often the reaction is something along the lines of "sounds great but it would not work in my class—with such an approach I could not cover the required volume of material." We are pleased to report, via this paper, that this is not necessarily the case. With some retraining on the part of the instructor, as well as the students, a significant initial investment on the part of the instructor, the incorporation of tools that support the learning process, and a CQI (continuous quality improvement) process in place, it does work.

1.0 Introduction

Undergraduate computer architecture courses have a large volume of material that students must learn. In addition, we want our students to develop written and oral communications skills, hone deductive reasoning and critical analysis skills, and be prepared to work in teams. To achieve these goals, it is necessary to employ a variety of learning strategies and tools, such as cooperative, active, problem-based, and team-based learning [2, 4, 1, 5, 9]. Although these strategies provide rich opportunities for a diverse student body to master the required course material and skills, they can be classroom-time intensive. This leads to challenges for both the instructor and the students. In general, the students must take on a more active role inside and outside the classroom and the instructor must carefully design the course, keeping in mind that the learning process is student-centric rather than instructor-centric.

Four significant challenges face an instructor when delivering a course using student-centric, team-based approaches.

1. When participatory learning strategies are employed, the quantity of material that can be covered during a class session is limited. Certainly, it is less than that which can be covered by a traditional lecture.
2. It is difficult to accurately assess individual mastery when assessing work products that are team generated.
3. Each student is different. It is necessary to adjust the course dynamically to meet the needs of the students in a particular class.
4. Some students need to be encouraged to utilize and develop critical thinking and analysis skills required in a student-centric course.

This paper focuses on ways to meet these challenges. After briefly outlining how cooperative, active, problem-based, and team-based learning are used in a first course in computer architecture, it concentrates on two specific approaches that we found to be particularly effective, Readiness Assessment Tests (RATs) [5, 9] and problem-based discovery learning. The remainder of the paper is organized as follows. Section 2 briefly describes the course in terms of content and our general philosophy towards delivery. Section 3 describes the structure of the course, RATs, and the laboratory exercises. RATs are used to motivate students to prepare for course activities. The laboratories use problem-based discovery learning techniques and are supported by both software and hardware tools. Section 4 reports on the results of both informal and formal assessments of the course and the learning strategies and tools employed. And, finally, in Section 5, we conclude with our view of our accomplishments and our plans for the future.

2.0 Course Philosophy and Content

The subject course, which is taken after a course in digital systems, provides students with an introduction to computer architecture and prepares them for a second, more advanced course in computer architecture, which uses the Patterson and Hennessy textbook entitled "Computer Organization and Design: The Hardware/Software Interface" [10]. The introduction to architecture and the prerequisite digital systems courses are intended to make the material in the first three chapters of the Patterson and Hennessy text accessible to the student without further assistance.

The mantra of the instructors, in this course as well as in their other courses, is attributed to Albert Einstein: "Make things as simple as possible, but not simpler." During this course, students peel away the layers of abstraction starting with high-level concepts and ultimately visualizing the execution of a program and controlling a robot via an assembler language program. Along the way, the instructor guides the students to discovery, feeding student curiosity as students deduce answers to questions.

Students enrolled in the subject course learn the general method by which a computer executes a program. They gain an understanding of basic computer architecture and work with assembler language and machine code. Related concepts covered in the course include the mapping of high-level programming constructs to low-level constructs, assembly and disassembly of instructions (instruction formats, addressing modes, effective address calculation), interpretation of Boolean algebra descriptions of instruction functionality, stored program concept (including, introduction to linking and loading), subroutine linkage, introduction to text, data, and stack segments, fetch/decode/execute process, machine state, evaluation of execution time (in cycles and time), and identification of exceptional behavior such as overflow. Additionally, students are introduced to I/O interfacing, interrupt handling, and robot control. They are reintroduced to basic concepts in software engineering including program design, implementation, modularization, testing, debugging, and documentation. The 68HC11 is used as a model processor architecture. Supporting materials include a textbook [6], a Motorola 68HC11 processor manual [7], and the software and hardware described in Sections 3.3.1 and 3.3.2.

The 68HC11 provides access to a 64KB address space (RAM) to store text, data, and stack, and four 16-bit and three 8-bit general-purpose registers. Memory-mapped I/O is used to access I/O ports, e.g., the analog-to-digital and serial communication ports. The ISA provides immediate, extended, direct, indexed, inherent, and relative addressing modes.

3.0 Course Structure

The course is scheduled for three hours of lecture and three hours of laboratory time per week. In-class sessions utilize a combination of learning techniques including cooperative, active, problem-based, and team-based learning [2, 4, 1, 5, 9]. After an introductory lecture (no longer than 10-12 minutes), students (formed into base groups, or sometimes groups formed in one of a variety of ways) are given a related task to accomplish, cooperatively, in a specified amount of time. While accomplishing the task, each group member adopts a different role, e.g., timekeeper, recorder, facilitator (ensuring participation by all group members), and questioner (ensuring that each group member understands and agrees with the group outcome). When the time expires, the instructor randomly selects a person from one group to describe her/his group's outcome. To add variety to class meetings, active learning [4], used on a one-on-one basis, also is employed.

Some of the virtues of cooperative learning (which is used in conjunction with problem-based and team-based learning) include honing of technical and communication skills via discussion and teaching, enhancement of critical analysis skills via discussion/debate, development of team skills, and growth of confidence (in particular, for students who feel more comfortable articulating the group's answer, rather than their own answer). Despite its benefits, when using cooperative learning, a facilitator must guide students' behavior (e.g., point out inappropriate behavior such as domination) and incorporate individual accountability so that all group members are held responsible for their own progress in the course.

In general, each major unit of the course is organized as a sequence similar to the following:

1. reading assignment (outside class)
2. RAT (in class, described next in Section 3.2)
3. lecture (in class, maximum 10-12 minutes)
4. in-class active/cooperative//problem-based/team-based learning using simple problems (described briefly above)
5. out-of-class problem solving using simple problems (cooperative and individual)
6. in-class active/cooperative//problem-based/team-based learning using more complex problems (described briefly above)
7. out-of-class problem solving using more complex problems (cooperative and individual)
8. simulator or robot lab (in lab, described in Section 3.3)
9. lecture (in class, maximum 10-12 minutes)
10. in-class active/cooperative/problem-based/team-based learning
11. assessment (for example, a quiz, which builds in individual accountability)

Depending on the targeted course material, steps 3 (optional), 4 and 5, and steps 6 and 7 may need to be repeated several times. Lectures or problem-solving sessions may need to be added to address hurdles encountered during RATs, problem-solving sessions, labs, quizzes, and exams; similarly steps 9 and 10 are used for this purpose. Individual accountability is built in to the course via weekly quizzes and three examinations, which are taken individually, and a final project, which is described in Section 3.3. Out-of-class problems can be worked on in groups or individually; one strategy is to alternate between the two.

The basis for a final grade depends on the instructor. An example of one used in the course is RATs 15%, quizzes 10%, exams 40% (a minimum average of 65 is required to pass the course), labs 10% (a minimum average of 60 is required to pass the course), homework 0% (optional; answer sheets are provided but homework is not graded), programming assignments 15%, and final project 10%.

3.1 Student Responsibility and Discovery Learning

Two major hurdles to increased student involvement in class are the difficulty in covering the required material in the limited time available and the need for students to utilize and develop critical thinking and analysis skills. To address the first issue, students must come to class prepared. In a traditional lecture setting, students achieve in-depth understanding of the subject matter outside of the classroom. Lectures introduce topics, and homework reinforces them—whether applying the concepts or preparing for a quiz or exam. Using the approach described in this paper, we attempt to reverse the in-class and out-of-class roles: the students introduce themselves to the topics by reading and working simple problems outside of class; in-depth understanding comes from applying their knowledge to more complex problems in and outside the classroom and in the lab, thus, utilizing and developing critical thinking and analysis skills. In class, the instructor observes students and becomes aware of difficulties that they encounter—this allows for intervention when necessary. The 12-minute lecture motivates the subsequent problem solving that will take place subsequently, explains a concept at a deeper level than would be possible without student preparation, and/or addresses issues that have proven (by observation in class or lab, or by performance on out-of-class problem solving, quizzes, or exams) to be difficult. Our techniques for motivating students to prepare for class activities are described in Section 3.2, while our strategies for encouraging critical thinking are exemplified in Section 3.3.

Readiness Assessment Tests [5, 9]. Using team-based learning [5, 9], a major instructional unit is addressed using the activity sequence depicted in Figure 1, which includes RATs as part of the readiness assessment process and is very similar to the sequence adopted in the subject course. A RAT is a quiz, taken immediately after a reading assignment; it tests whether or not a student did (with a reasonable amount of depth) the assigned reading. The assigned reading, in turn, prepares students for a class session that is meant to enhance the learning accomplished via the reading. The questions are true/false and/or multiple-choice. A sample question might be “Overflow cannot occur when the numbers are not of the same sign. True or False?” An inappropriate question is one that requires the understanding of a key concept, for example, a multiple-choice question regarding how overflow is detected. Save these types of questions for quizzes!

First, a RAT is taken individually; students record their answers prior to handing in the RAT. Immediately thereafter, the same RAT is taken as a team. All RATs are taken by the same "base" team. Base teams are assigned for the duration of the course. (Here the term “team” is used, rather than “group”, because the intent is that the conditions created during the life of the course will turn the base “group” into a “team”.) After the first RAT, the student body comes to consensus on how much the individual and team RATs are to be

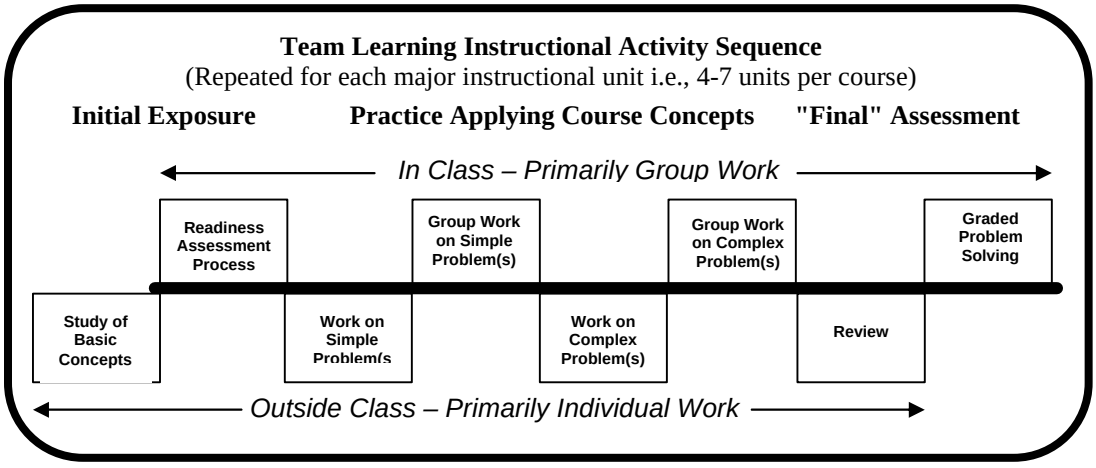

Figure 1. Team learning instructional activity sequence.

3.2 Motivating Outside Preparation

Given the best effort of an instructor with respect to designing an effective, participatory class session, it is doomed to failure without adequate student preparation. A technique that addresses this problem is RATs, weighed w.r.t. each student’s grade. For example, the individual RAT score might be worth 40% of a student’s score, while the team RAT is worth 60%. Of course, the instructor can set a threshold. Scantron tests make this whole process easier. If a team does not agree with the grading of a question, they are permitted to submit a written appeal [5, 9], which the instructor evaluates. If the instructor agrees with the appeal, then only that team receives an adjusted score—this rewards critical analysis and effective communication. Of

course, if the instructor's answer is incorrect, all teams' RATs are regarded.

We have observed that RATs are effective in several ways. RAT scores were generally high; for example, in spring 2002, 62% of students averaged 80% or higher on the combined RATs. The combined RAT grades are averaged from the grades obtained in the individual and team RATs. This and the fact that discussions during team RATs, which are animated and passionate, indicate that the vast majority of students do, indeed, read the assigned material. Students seem to cover a sufficient amount of material on their own and are better prepared for collaborative, active, problem-based, and team-based learning. The taking of a RAT as a team incorporates many of the virtues of collaborative learning.

3.3 Supporting Discovery Learning

The laboratory sessions, which are based on those described in [11, 12], focus on problem-based discovery learning. This approach supports the learning process in several ways. It is used to introduce, explore, and strengthen understanding of new concepts. Simulator- and robot-based laboratory assignments (labs) guide learning via different levels of abstraction. The well-defined, proctored, cooperative learning labs meet twice a week. They are experimental labs, similar to labs in physics or chemistry, that are designed to lead students down a path of discovery, affording them opportunities to teach themselves and others via the scientific method. Students work in pairs using, for example, brain storming to decide on an approach to use to discover an answer or to understand the results of an experiment.

The goal of each lab is stated explicitly, and the lab is written in such a way that students are required to experiment, critically analyze experimental results, and use deductive reasoning to arrive at correct answers. Working in pairs, and sometimes in larger groups, reinforces their confidence in their understanding of jointly constructed hypotheses, experimental designs, and predictions, their analysis of experimental results, and their understanding of the concepts that underlie the lab. Of course, this also helps improve their communication and team skills as well.

This course uses two types of tools in labs: a simulator and robots. Simulator labs focus on basic machine organization and assembler language using the Visual 6811 simulator [8]. Robot labs focus on memory-mapped I/O, serial communication, interrupt handling, and I/O interfacing. A lab sheet, which describes the goals of the lab and what is expected of the students, is presented to students prior to a lab; the deliverable of each lab is either a completed lab sheet (that includes answers to questions, as described below) or a demonstration of a working program. Students unable to complete the lab during the allotted time must finish it on their own time; this is a typical situation for robot labs, which anticipate complex problem solving being done outside of the labs. Simulator and robot labs, and the tools they employ, are described in the following sections.

3.3.1 Simulator Labs and Visual 6811

Simulator labs are carried out in pairs by writing and executing assembler language programs, examining assembler listings, symbol tables, and cross-reference tables, and simulating the execution of an assembler language program using the Visual 6811 simulator, a simulator for the Motorola HC6811 microprocessor. Each simulator lab brings together different pairs of students. This permits students to get to know other students in the class and provides challenges with respect to communication and team skills.

A simulation permits the examination of register and memory contents including the text, data, and stack segments. The simulator also displays the execution time in cycles. Via simulator labs, students can analyze, among other things:

- representation of instructions (in machine code),
- initialized and uninitialized data,
- instruction length and instruction format differences attributable to addressing modes,
- effects of instruction execution,
- effective address formation,
- program control flow,
- behavior of subroutine calls and return instructions, and their effect on the runtime stack,
- composition of activation records,
- differences between call-by-value and call-by-reference semantics,
- contents of interrupts vectors, and
- algorithmic complexity in terms of execution time.

Visual 6811 is composed of two independent subsystems that work together to provide the functionality to debug a program and to simulate the execution environment of a program written in the 68HC11 assembler language. The two subsystems are a simulation engine, which is composed of a library of functions, and a GUI, which controls the simulation of the program.

The simulation engine can simulate the complete 68HC11 ISA. (Note, however, that some underlying subsystems such as the asynchronous serial communication interface and interrupts are not yet implemented.) Simulation is at the instruction level, as opposed to the cycle level. The library provides access via the GUI to fetch and modify simulated components such as registers and memory.

The GUI controls the execution of a simulation, displays changes to the execution environment of a program as it executes, and provides debugging functionality. The GUI subsystem is platform independent; Visual 6811 can run on any system running the Windows, Unix, or Linux operating systems and that supports the Java Runtime Environment (JRE). Figure 2 shows an image of Visual 6811 while simulating a program. The image shows a breakpoint that is set (instruction highlighted in red) and the next instruction to be simulated (instruction highlighted in cyan). After a program is assembled, the GUI displays the assembler listing as shown in the left side (circle 1) of Figure 2. The assembler listing is composed of the program's assembler instructions and the corresponding machine code.

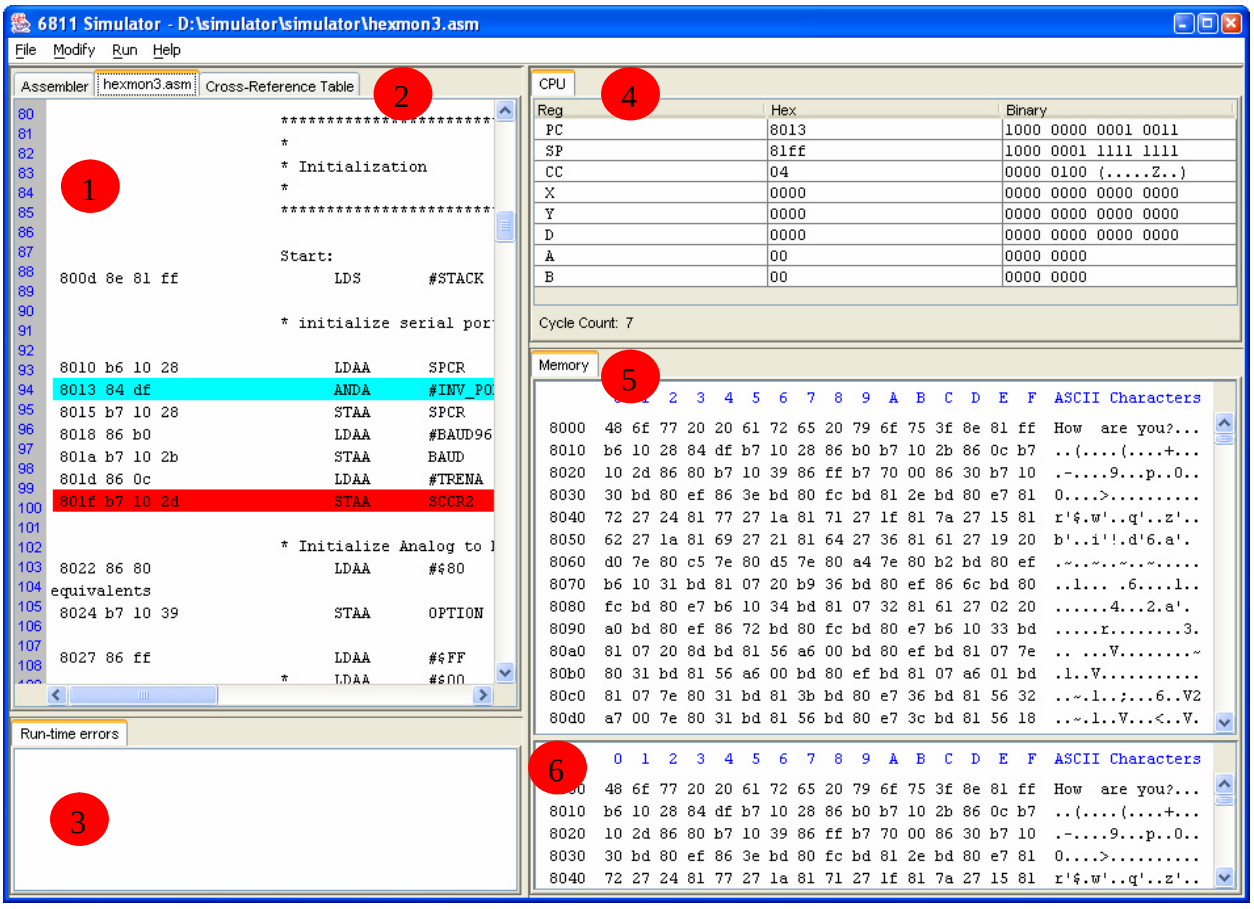

Figure 2. Visual 6811 while simulating a program.

By selecting a tab (circle 2 of Figure 2), the cross-reference table can be displayed. The lower left part (circle 3) of Figure 2 shows the run-time error display. An error is generated, for instance, if a bad opcode is fetched. The upper right section (circle 4) of the simulator window displays the names of registers and their contents, both in hexadecimal and binary. As a program is simulated, register contents are updated in response to the (simulated) execution of instructions. The right part of the simulator interface shows two windows (circles 5 and 6) for displaying the contents of memory. The memory is laid out to show 16-byte memory blocks. Each block is displayed by showing the starting address of the block, the hexadecimal value of the memory, and the ASCII representation. Students may view two areas of memory, one in each of the memory windows. This allows students to view the data and stack segments simultaneously. Instead of displaying the stack in its usual abstract notation (a stack of values with the most recent value stored at the top and the oldest value stored at the bottom), the stack is displayed in the same format as memory to solidify the notion that the stack is just another area of memory.

Visual 6811 provides the following features to debug and control the simulation of a program:

- *assemble a program*: Assemble the selected program and display the assembler listing. After this step, the program can be simulated.
- *reassemble last assembled program*: Reassemble the last file that was assembled (successfully or not).

- *modify memory contents*: Modify the contents of memory by entering a new value to store at a specified memory address.
- *modify register contents*: Modify the contents of any of the simulated general-purpose registers. This and the previous option make it easy to modify the execution environment of a program while it is simulated and quickly see the execution behavior that results from these changes.
- *step through the execution of a program*: Simulate a program one instruction at a time, permitting the observation of changes in the execution environment as a result of instruction execution.
- *run a program to completion*: Execute a program to completion, as opposed to one instruction at a time. This is most useful when breakpoints are set.
- *toggle breakpoints*: Set or clear breakpoints in a program. This is useful when debugging large programs and when analyzing specific code sections.
- *stop a simulation*: Interrupt a simulation that is running to completion. This is useful, for instance, if a program has entered in an infinite loop, so the only way to get out of the loop is by stopping the simulation.
- *reset execution of a program*. Restart a simulation by loading into memory the machine code of the last program assembled.

3.3.2 Robot Labs

After acquiring some competence at programming the 68HC11, students are challenged to program small robots that are controlled by 68HC11s. The TJPro robots [3], depicted in Figure 3, have infrared (IR) emitters and detectors, and bumper sensors, expanding the possibilities for programming the different interfaces, such as the analog to digital interfaces. Robot labs require students to download programs, via the serial communication interface, to a 68HC11 microprocessor. These programs interface with the 68HC11's I/O ports and permit students to analyze the behavior of, among other things:

- memory-mapped I/O,
- the serial communication interface,
- motors connected to the I/O ports,
- digital and analog sensors,
- programmable timers and counters, and
- interrupts.

The robot labs allow students, paired into fixed teams of two (which are formed based on students' input re: their top six choices), to leap from abstract concepts and simulations to a more hands-on experience. In addition, they permit students to become involved in fun and challenging projects. For instance, the first robot lab has students use the serial communication interface to display the contents of the robot's memory (the 68HC11) on the monitor of the host PC. Pressing a key on the host's keyboard, which instructs the 68HC11 to read from or write to a memory byte or word, drives the related program.

Final projects have been very successful in challenging students to incorporate in a program all the knowledge they have acquired throughout the semester. The first final project had students program robots to navigate a maze. To make this more challenging, not only did the robots have to find their way out of the maze, they had to remember the path that they followed so that when the robots were next placed at the beginning of the maze, they navigated the maze without bumping into walls. Another challenging final project was to program two robots equipped with IR detectors: a *wimp* and a *follower*. The *follower*'s goal was to detect the *wimp* and tap it. The *wimp*'s goal was to avoid being detected by the *follower* and exit the arena before getting tapped by the *follower*.

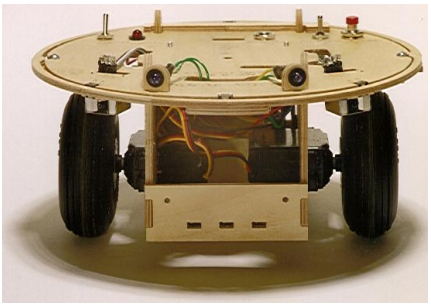

Figure 3. TJPro robot.

Individual accountability is achieved by defining projects that (1) are comprised of multiple interfacing functional components, which when complete meet the project specifications, and (2) have multiple subgoals that can be achieved by a subset of complete functional components. In this way, if one team member does not do her/his part, the other team member does not get penalized in so far as the project grade is concerned. Individual accountability also can be enforced by having individual students present the results of the team effort.

Besides all the benefits attributable to collaborative, problem-based discovery learning, the robot labs require students to operate at a higher level, especially with respect to debugging. Considering debugging techniques as a continuum ranging from random (uneducated, uninformed), to brute force (uneducated, informed), to guessing (educated, uniformed), and finally to experimental (educated, informed), the complexity and difficulty of robot-based labs make the first three forms of debugging ineffective. This leaves the students with only one viable alternative: an experimental, analytical approach. As a result, the student must

- be clever about creating a hypothesis,
- be resourceful in testing hypotheses,
- discuss hypotheses in terms of observed behavior, not the underlying computer architecture, and
- discuss solutions in terms of the underlying computer architecture.

Just what we want!

4.0 Assessment

The strategies and tools used in this course have been assessed both informally and formally. The informal assessment concentrates more on the learning strategies employed in the course, while the formal assessment focuses on Visual 6811, the simulator used in the course. In addition, student evaluations from fall 2001 and spring 2002 indicate that the course was well received.

A 14-question informal assessment tool was used the second time the course was taught in this way (in spring 2002) to assess the effectiveness of the learning strategies and the simulator and robot labs employed in the course; the course has been taught this way twice a year since fall 2001. In general, the students (27 in number) were most positive about discovery learning via the simulator and robots. Opinions were divided with respect to working in groups during class and labs. On the positive side was the opportunity to share ideas, opinions, and reach some conclusions; on the negative side was the frustration experienced as a result of unprepared group members, who might affect their grades. Weekly quizzes were strongly favored—students stated that quizzes help them study for quizzes and exams (especially when they include questions that focus on the same concepts), and keep concepts they learn fresh in their minds. On the other hand, some students thought quizzes increased stress. The majority of students thought it was a good idea to use the robots in the course. Programming the robots helped to reinforce the concepts they learned and to see (physically) the power of the 68HC11 architecture in action. Some interesting student comments follow.

With respect to the simulator:

- "[The simulator] helped me achieve knowledge about the material covered in the class."
- "[It] helped me to understand what was happening."
- "We could test and see what happened for ourselves."
- "[It] helped me learn how to assemble and disassemble."
- "[It] helped me with programming."

With respect to the robots:

- [The introduction of the robots] was a good idea."
- "[They] enforced the concepts learned in class."
- "[They] gave me confidence that I can do something that seems impossible at first."
- "[They] were fun!"
- "[They] made it possible to see the power of the 68HC11 in action."

Recently a formal assessment, which focuses on Visual 6811, was conducted. The assessment is based on the response of 25 students, who either just took the course this semester (spring 2003) or who took it in the past two years. The assessment indicates the following:

- The simulator was straightforward to use and had a very low learning curve (23 students, i.e., 23/25).
- Being able to view the assembler source and the virtual representation of memory was helpful in understanding how memory changes as a program is executed (19/25).
- The simulator was particularly helpful for understanding stack manipulations and the differences among the various addressing modes (22/25).
- The simulator facilitated debugging by permitting the user to step through program execution and, while doing so, observe how memory and registers changed (18/25).
- Given the choice of using the simulator or not, students said they would have chosen to use the simulator since it helped them to debug and understand the execution of a program (21/25).

In fall 2001/spring 2002, of 24/19 responses to a student evaluation, out of a 1-5 rating, where 5 is excellent:

- The varied use of questions, discussions, lectures, and/or group work in the class was rated 4.8/4.9 (average rating).
- The relevance of course materials to state course objectives was 4.4/4.8.
- The relevance of class assignments was 4.5/4.7.
- The estimation of how much learned in the course was 4.4/4.6.
- The effectiveness of the course in challenging the student intellectually was 4.7/4.7.
- The overall rating of the course was 4.5/ 4.7.

5.0 Conclusions and Future Work

More powerful assessment is needed in order to quantify the success of the learning strategies and tools used in the course. But if student evaluations, student comments, and instructor observations are considered, the course indeed has been effective. In our opinion, the most powerful of the learning strategies employed in

the course are the RATs and the cooperative, problem-based discovery labs (both simulator and robot). In addition, it cannot be denied that collaborative, active, problem-based, and team-based learning, i.e., learning that actively involves students, is far superior to lecture.

The next step in the development of the course is to extend the capabilities of Visual 6811 to simulate both the functionality of the serial communications interface and interrupt handling. In the very long run and with sufficient interest, a textbook that incorporates both the technical information conveyed via this course and the strategies and tools used as the vehicles of conveyance is envisioned.

6.0 References

[1] Davis, B. *Tools for Teaching*, San Francisco: Jossey-Bass, 1993.

[2] Johnson, D. W., R. T. Johnson, and E. J. Holubec, *Cooperation in the Classroom*, Edina, MN: Interaction Book Company, 1992.

[3] http://www.mekatronix.com/

[4] Meyers, C., and T. B. Jones, *Promoting Active Learning*, San Francisco, Jossey-Bass Publishers, 1993.

[5] Michaelsen, L. K., and R. H. Black, "Building Learning Teams: The Key to Harnessing the Power of Small Groups in Higher Education," in *Collaborative Learning: A Sourcebook for Higher Education*, Vol. 2, S. Kadel and J. Keehner (Eds.), State College PA: National Center for Teaching, Learning, and Assessment, pp. 65-81, 1994.

[6] Miller, G. H., *Microcomputer Engineering*, Upper Saddle River, NJ: Prentice Hall, 1999.

[7] Motorola M68HC11 Reference Manual, 2001.

[8] Nieto, Manuel, *Visual 6811: A GUI for Simulation of the Motorola 68HC11 Microprocessor Architecture*, Master's thesis, University of Texas at El Paso, Department of Computer Science, 2003.

[9] www.ou.edu/idp/teamlearning/

[10] Patterson. D. A., and J. L. Hennessy, *Computer Organization and Design: The Hardware /Software Interface*, San Francisco, CA: Morgan Kaufman Publishers, 1997.

[11] Teller, P., "Experimental, Cooperative Labs in a First Course in Computer Architecture," *Proceedings of the 1997 Frontiers in Education Conference (FIE '97)*, Pittsburgh, PA, CD-ROM, November 1997.

[12] Teller, P., and T. Dunning, "Mobil Robots Teach Machine Programming and Organization," *Proceedings of Supercomputing `95*, San Diego, CA, CD-ROM, December 1995.