Integrating Assembly Language Programming into High School STEM Education (Works in Progress)

Mr. Joseph Foy, L&N STEM Academy, Knox County Schools

Joseph Foy holds two degrees in Electrical Engineering; BSEE 1976, U.S. Naval Academy and MSEE 1988, University Of Tennessee. His twenty-four year engineering career included responsibilities of programming, gate array design, hardware design, field service, and manufacturing support. For the last seven years, Mr. Foy has been a high school teacher in Knoxville, Tennessee. He is licensed to teach all high school math and physics courses. In 2011 and 2012, he was selected for the CURENT RET (Research Experience for Teachers) program, and in 2013 for an REV (Research Experience for Veterans) program. At CURENT, Mr. Foy developed curriculum materials which assist him in explaining power generation and transmission to high school math, physics, and programming courses.

Dr. Chien-fei Chen, University of Tennessee, Knoxville

Chien-fei Chen received the B.S. degree in English Language and Literature from National Cheng Kung University, Taiwan, in 1992, and the M.S. in Communication, and Ph.D. in Sociology degrees from Washington State University in 1995 and 2009, respectively. Her current research interests include public acceptance of smart grid, renewable energy and energy conservation, and engineering education (K-12 and university). She is a research professor and co-director of education and diversity program at NSF-DOE engineering research center, CURENT and an adjunct faculty in the Department of Sociology at UTK. Prior to her academic career, she worked in the media industry including KSPS -Spokane Public Station, KCTS-Seattle Public Television, Seattle Chinese Television Station, Public Television Service, Taipei, Vision Communication Public Relation Company, Taipei. She was also a research scientist at Virginia Tech and lab manager at Washington State University.

Mr. Erin James Wills, University of Tennessee, Knoxville

Erin Wills received a B.S. in Chemical Engineering from Purdue University and a M.S. in Theory and Practice in Teacher Education from the University of Tennessee. He currently works as the education coordinator for the Center for Ultra-wide-area Resilient Electric Energy Transmission Networks (CURENT), a NSF/DOE engineering research center. His role at CURENT involves developing K-12 curriculum related to electricity, renewable energy, and the electric grid for the center's pre-college outreach programs. In addition, Mr. Wills serves the undergraduate and graduate electrical engineering program at the University of Tennessee by arranging research and industry specific training for the students. Prior to joining CURENT, he worked as an operations engineer and automation engineer in the chemical industry.

Integrating Assembly Language Programming into High School STEM Education (Works in Progress)

Abstract

Learning assembly language as an introduction to programming can be beneficial to high school students. From the very beginning, students are taught programming relationships to a processor through the use of registers, memory, and the control unit. Students can build upon these fundamental ideas so that higher level languages are more thoroughly understood. This paper describes a high school STEM education curriculum that provided sophomores hands-on opportunities to learn and understand microcontrollers through assembly language projects. The course assessment evaluated the students' computer science knowledge, course expectations, learning perspectives, creativity, and future field of study interests. Initial results indicate that students have a greater breadth of knowledge, a stronger positive perception of computer science, and a greater self-efficacy while at least maintaining student interest and creativity. Observations of the students indicate that the investigative nature of programming with microcontrollers is motivating the students to seek further programming courses.

I. Introduction

Although the integration of programmed technology has become more common in everyday products, very few secondary schools offer computer science courses to their students[8]. Typically, Advanced Placement (AP) Computer Science[7] is the only course offered to introduce programming to students despite a high demand in the job market for computer science related degrees[5,8,12]. Of the millions of students in the United States, only about 25,000 students took the AP Computer Science exam[6]. To even complicate the problem further, computer science, if offered, is typically presented to students as programming despite the discipline being about managing and processing information[2]. These factors can distort students' perception of what computer science is and the opportunities available.

The courses that are offered in schools often emphasize the manipulation of data and do not include interfaces with hardware. Vahid and Givargas identified that most programming in early years is data oriented instead of time-oriented[14]. From their experiences, they suggested that time-oriented programming be introduced to students at an early age so that students develop dual understandings as they learn higher level languages[14]. Otherwise students may develop a conceptual model of higher level programming that conflicts with their understanding when they begin to learn about low level programming. Students have been observed having difficulty when learning about C language if their only background is in Java, but students with some assembly background have transitioned much better[15].

Many current methods of teaching programming have involved avoiding assembly language because it has a reputation as being difficult for students to understand, but doing so may reduce students understanding of the fundamental operations of the processor[4]. Instead of discarding the use of the language, alternative methods of teaching may be necessary like breaking projects into smaller pieces[4], creating concrete and measureable goals[15], and avoiding unnecessarily

complicated operations of assembly[11]. Overall, a balance is needed between assembly language and higher level languages when a student begins learning to program.

Currently, the relationship between hardware and software is mostly reserved to non-introductory computer or electrical engineering courses at the collegiate level. Due to the commonplace use of microcontrollers in applications[1] and the task selective nature of choosing the appropriate language, it can be argued that assembly and a higher level languages should be taught[13]. In addition, some studies have found that understanding programming as it relates to hardware to be less effective when only high level languages are used[9]. Due to these reasons, a high school assembly language course was designed as the first programing experience for students who are interested in programming.

The class was structured in similar ways to some college courses that also incorporated assembly language as a short course or precursor to using higher-level languages[10,16]. As with the college courses, students were using a microcontroller from the beginning of the semester and learning the programming process through hands-on modules[2,12]. Other similarities included using prefabricated microcontroller boards due to time restraints[1,10,16] and gradually increasing the complexity of the technical content[9]. In the end, the students may be better prepared with conceptual understandings of computer science. Some research has supported the idea that programming should begin by learning assembly and then moving to higher level language such as C[1,10].

This paper is organized into four additional sections. Part II describes the microcontroller, course schedule, and the instructor goals. Part III provides a more detailed summary of the three initially designed labs, explains the effect of the initial labs, and describes two new labs that were implemented. Part IV analyzes the knowledge and perspectives that the students possessed. Lastly, Part V reviews the future of the assembly language course.

II. Course Organization

A. Materials

We have designed and delivered an assembly language programming course to sophomore high school students. During the fall semester of 2013, eight students voluntarily selected to learn to program Microchip Technology's PIC™ 16F616 microcontroller.

The PICDEM starter kit was selected since it contains the PIC16F616 microcontroller, but several other microcontrollers are included in the kit. The PIC16F616 was selected specifically since it is an 8-bit controller. Although most controllers found in smartphones are 32-bit ARM processors, the less powerful processors have a smaller instructional set, are less costly, and contain only the necessary functions to learn fundamental processor architecture. The PIC16F616 has a limited set of features that include memory, timers, analog-to-digital converter, and pulse width modulators. Besides the microcontroller, the PICDEM development board includes a DC motor, oscillator, speaker, potentiometer, bread board, and other components as seen in Figure 1. The code was developed through the MPLAB Integrated Development

Environment (IDE) programming interface with frequent references to Microchip documentation.

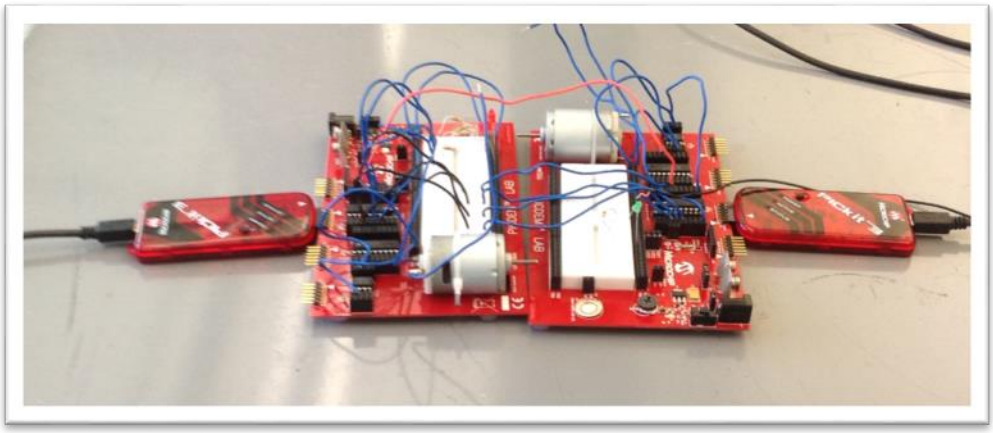

Figure 1: Two PICDEM boards used together for Lab 3

Standard computers with a USB port are capable of running the IDE. For this class, students used MPLAB IDE on MacBook Airs to program. The USB port is used to power and transfer code developed on the computer to the microcontroller. External power sources can also be used.

B. Schedule

Initially, the course was designed to use five 90 minute lectures to cover main topics followed by three labs requiring two to three 90 minute blocks. Lectures included topics regarding parts of a microcontroller, detailed information about the PIC16F616 microcontroller, and usage of assembly language. A separate lecture was usually required at the beginning each lab. The Microchip manuals were used during coding since these guides were appropriate complexity for the students. The school operates on a 90-minute block schedule where the class meets every-other day. Table 1 provides an overview of the topics and schedule that were initially designed. Due to interruptions and adjustments for students' needs, the schedule was extended (primarily for Lab 3).

Table 1 –Initial Schedule

Phase	Schedule	Topic
Introductory Lectures	Day 1	Microcontroller Overview - Development Board, embedded control
	Day 2	Introduction to PIC16F616 microcontroller. Topics include memory map, registers for first lab (Timer0, Timer1, general purpose input/output (GPIO) ports)
	Day 3	Introduction to specific commands from the PIC16F616 instructional set. Topics include command syntax, working register, data movement, how to initialize registers
	Day 4	Exposure to a Sample Program. Instructor showed students sequences of commands and basic program divisions from a pre-written program
	Day 5	Installation of the Integrated Development Environment (IDE)
LAB1 Basic Assembly	Day 6	Writing a basic program
	Day 7-9	Instruction for Lab 1: Topics include GPIO, assembling breadboard area, downloading code, turning on LEDs Goal: Students change code to create a new LED lighting pattern
LAB2 Interrupts	Day 10	Review lessons learned from Lab 1 Instruction for Lab 2: Topics include concept of variable, interrupts, and ISR (Interrupt service routine)
	Day 11-14	Lab 2 Goal: Students create an interrupt from an internal timer that will cause a stored variable to be incremented to change the LED lighting pattern
LAB3 External Interrupt	Day 15	Review lessons learned from Lab 2 Instruction for Lab 3: Topics include data strobe and external interrupt
	Day 16-19*	Lab 3 Goal: Students send 4-bits of parallel data via a data strobe line from the microcontroller to a second microcontroller that reads the data. The receiving controller's code interprets the data as an external interrupt and writes the parallel pattern to the LEDs.

* Lab 3 was extended by multiple days due to technical issues

C. Teaching Approach

The initial plans for introducing assembly language to high school students was to begin with very simple projects and slowly introduce code for the students to analyze and understand the organization and syntax. As processes and vocabulary became routine, the complexity of tasks increased. The projects were designed to use previous sections of code that the students were familiar using to help develop a strong understanding of what was happening and how the different parts of the code functioned. It was important not to overload the students with too much information or responsibility for self-learning at the beginning. With each lab, students developed troubleshooting skills that could be applied to the next lab, and self-learning could have a greater role. Figure 2 illustrates the course topics and goals.

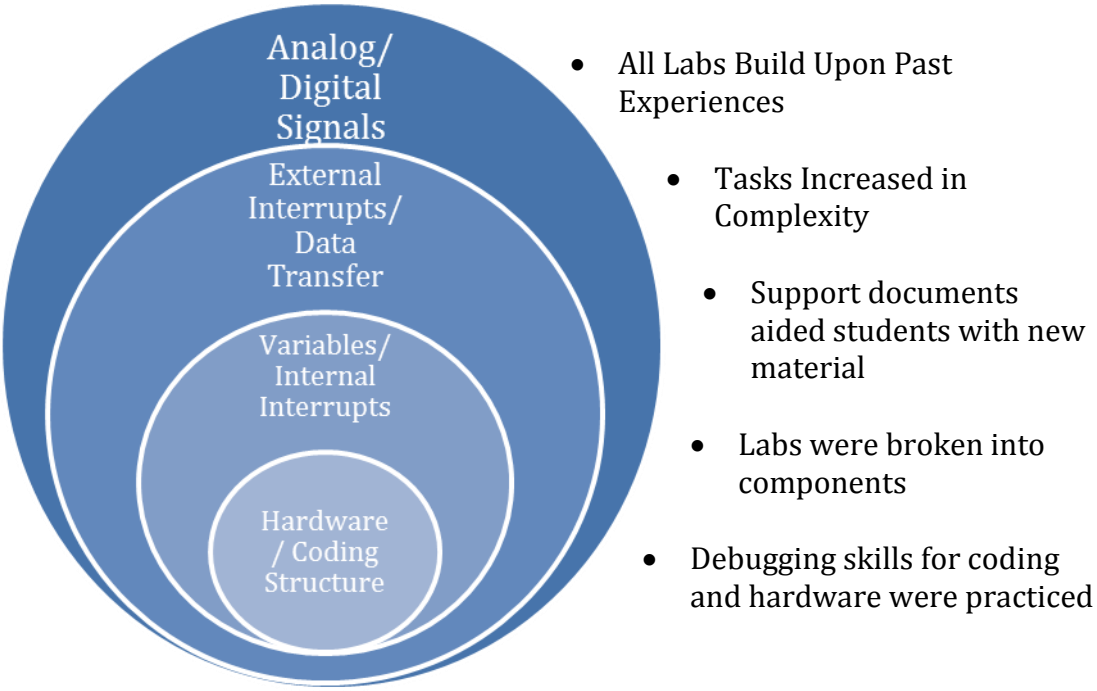

Figure 2: Learning Construction and Increased Complexity

III. Lab Activities & Lab Review

A. Initial Labs

The first lab introduced students to basic tasks such as using the IDE to enter code. The instructor guided the students through the coding exercise and provided explanations regarding how each line of code was related to the internal architecture of the microcontroller. For example, the students learned that a microcontroller sequentially reads a specific memory location that can contain a section of code before moving to the next memory location that can contain additional code. The first program allowed the students to control a set of LED lights and consisted of two sections called the INIT (initialization) and the MAIN (continually running code). Although the program could be written with only a MAIN section, the program was divided into two sections so that the structure of future labs, which require an INIT section, would be consistent throughout the entire course. The INIT was used to configure six outputs that were connected to resistor-LED combinations. The MAIN for the first lab contained instructions (binary output) to control the LED lights.

The second lab kept the same coding and hardware structure but introduced a new section in the code called the Interrupt Service Routine (ISR). This program allowed the students to control the LED pattern by using a timer and variable. In the INIT, the students learned how to configure a (16 bit) internal timer, TIMER1, which generates an interrupt. The ISR coding section was mapped to a specific memory location which was similar to the concepts used in the first lab when the students identified the location of the INIT. The MAIN section remained the same except that students learned how to define a variable called “count” and increment the variable so that a new pattern could be passed through output pins to LEDs. The final stage of

the MAIN was to clear the TIMER1 interrupt. Ultimately, the program operation was being guided by the ISR section. The partition of control between the ISR and the MAIN will assist students should they take an advanced programming course (AP Computer Science A) offered at the same high school. In courses that use an object oriented programming language like Java, programming terms called events and actions are very analogous to the assembly concepts of interrupts.

The third lab required groups of students to work together to create two sets of interacting code. Two development boards were wired so that output pins of one microcontroller were connected to the input pins of the other microcontroller. One group of students called the "write group" developed code that would send an incrementing value and a signal indicating the value has updated to the other microcontroller. The second group called the "read group" developed code that waited for the signal from the other microcontroller and then displayed the value. The value is a 4-bit incrementing count signal that updates based upon the TIMER1 interrupt. The additional signal is often called a "data strobe" or "read\write line" by engineers. The sequence for the write group required the students to retrieve the value of the variable "count", increment it, store it in memory, write it to output pins and then trigger the fifth pin to change from logic "low" to logic "high" and then back to "low." The sequence for the read group required the group to connect four of the pins that were data pins and the "data strobe" pin from the "write microcontroller" The data strobe was configured to be an external interrupt in their program.

B. Lab Review

After completing the first three labs, it became apparent that additional lectures and support were needed for the students to advance their learning. Therefore, a simple set of scaffolds was added to the existing curriculum. At the beginning of each new lab, an introductory lesson was provided to introduce new hardware and software concepts and time was given to practice these new concepts. In addition, having the students do a short code analysis of the new concepts was beneficial. Lastly, showing the students how to debug their code proved to be extremely valuable. Debugging guides, trialing pre-tested code, and testing the code in sections helped the students troubleshoot.

A common problem of the labs was managing time so several changes are being made to improve the course pacing. For example, time was lost debugging student work while labs were being tested with two microcontrollers. To correct this problem a simulator will be used to troubleshoot initial problems. Additionally, some technical changes will be implemented after experiencing problems during the labs. One problem occurred when two boards that were connected together created a voltage difference across the boards which caused erratic behavior. A standalone power supply may help reduce this problem or a different method of connecting the boards may be needed. These options will be explored before the next trial of the lab, but additional improvements will ultimately be found as the lab is used each semester.

The lessons learned from the first three labs included placing greater emphasis on introducing a lesson by explaining new material in greater detail and practicing the concept where applicable, analyzing the code and using a debugging guide to trouble shoot problems, and pretesting

hardware and using debugging statements in the code to identify problem areas. Figure 3 shows the general process for coding used in this course.

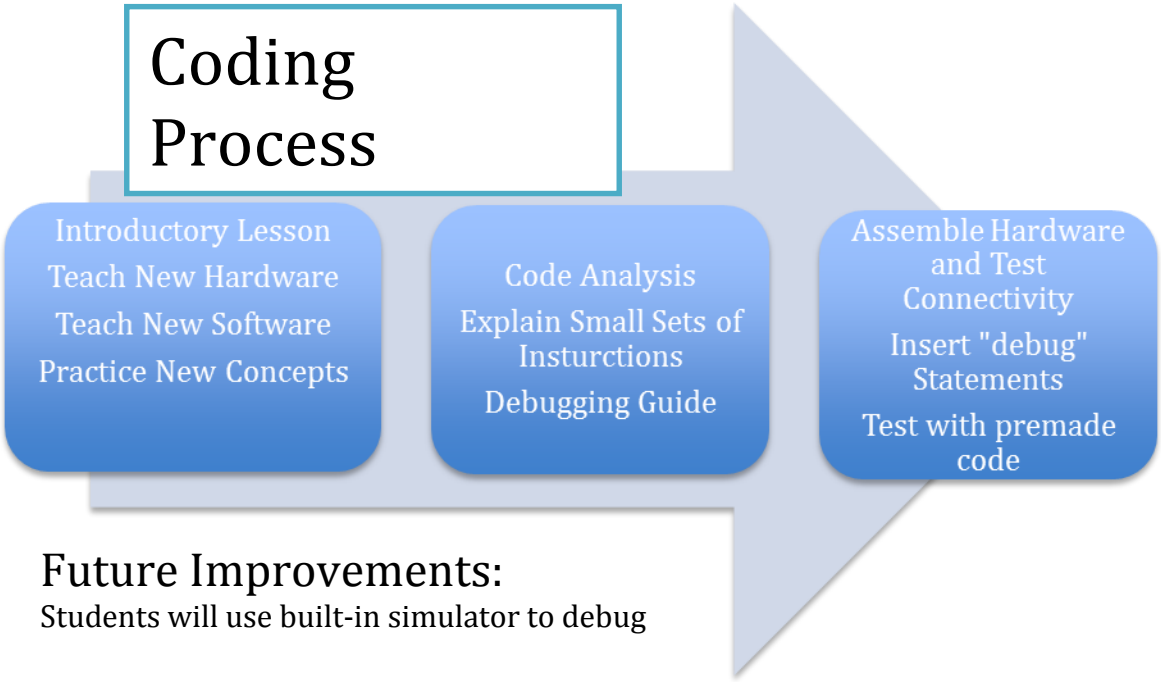

Figure 3: Steps to developing working code on microcontroller

Besides additional scaffolds to help students successfully navigate hardware and software issues, several lecture topics were identified as needing extra attention. Topics including hexadecimal, binary, decimal, numerical conversions, program counters, stacks, digital and analog signals, IDE functionality, test instruments, software and hardware debugging, and simulation were areas that the students needed extra support.

With time remaining in the course and positive feedback from the students, two additional labs were developed. The sequence developed during labs one through three seems effective, as demonstrated by the rapid and efficient execution of labs 4 and 5, which were guided by the sequence in Figure 3. The fourth lab involved passing an analog value to the microcontroller so that it could be converted to a digital value and displayed on an external LED. The fifth lab used three interrupts to initiate different LED lighting patterns. The final labs operated much better than the initial labs. The following section describes specific activities for each new lab.

C. Modified Labs

The fourth lab used an internal Analog-to-Digital Converter (ADC). Lectures explained analog and digital signals, and how one represented the other. A potentiometer on the breadboard provided the analog input signal and the students configured the ADC in their INIT routine and initiated the ADC conversion in the ISR routine by a timer interrupt. The reading and the writing to the LED lights did not occur until the code initiated in the ADC conversion and issued a completion bit. If the completion bit indicated that the ADC conversion was not complete, the

program returned to the checking instruction. When the conversion bit indicated completion, the program skipped the "return and check" instruction, and instead proceeded to read\write values. An important concept and assembly instruction was introduced – the concept of a logical decision.

The fifth lab used three interrupts, TIMER1, and two external interrupts which was very similar to the strobes used in Lab 3. For this lab, students created a more advanced logic system. When an interrupt occurred, the program needed to determine the source. This involved several sets of the logic instruction used in Lab 4. The timer interrupt caused the variable "count" to be incremented and written to output LEDs. One external interrupt caused the LED pattern to slow down by decreasing the timing rate. The second external interrupt caused the LED pattern to speed up by increasing the timing rate.

IV. Course Assessment

A. Course Assessment

An online post survey was chosen for assessing the impact after the course was completed. The surveys instructed the students to read a series of statements and rate each statement based on a 5-point Likert-type scale (1 = *strongly disagree*, 3 = *neutral*, and 5 = *strongly agree*). The survey instrument was designed to measure the following concepts:

- course expectations
- knowledge in computer programming
- interest in learning computer science
- perception of computer science
- confidence in learning STEM
- confidence in learning specific tasks related to programming
- attitude towards programming
- creativity and future career choice

Specifically, *course expectations* were measured by four statements including the course is expected to be challenging, to help students better understand programming, to encourage students to pursue a computer science major and to help them decide if computer science is the right field for them, and to connect students with similar interests. Three statements regarding students' familiarity with general computer science concepts, basic computer language and application of a computer language measured *computer knowledge*. *Interest* was measured by three statements to assess students' interest in learning computer science in general, in learning a specific language, and in learning computer science as a potential career. *Perception* of computer science was measured by six statements including computer programming allows people to do more interesting work, to do more imaginative work, to enlarge individuals' scope, to help solve problems, to make greater contributions, and to acquire relevant information that students need. *Confidence in learning STEM* was measured by five statements including the belief that the student is able to learn engineering/science concepts well, learn programming well, discuss engineering/science ideas well, carry out a research project well and solve problems in science or engineering topics effectively. *Confidence in computer science* was measured by 18 statements. Examples of those statements include the belief that a student can learn to use a

variety of programs, write simple programs and explain why a program will or will not run. *Attitude towards computer science* was measured by four statements regarding student interest toward programming as appealing, fascinating, interesting and meaning a lot to the students. *Creativity* was measured by five statements including the belief by students of their ability to think creatively about programming, to try out new ideas on their own, to work extra hours because they are excited about programming and to take risks in programming without being afraid.

B. Survey Results

Overall, the seven students who enrolled in this course were very satisfied (57.1%) or somewhat satisfied (28.6%) with the course and no one reported they were dissatisfied with the course. In addition, 71.1% of the students indicated this course has met their expectation and 71.1% of the students reported that this course was important in their decision to studying engineering or CS in college. 85.8% of the students would recommend this course to their friends. The majority of the students (85.8%) reported that they would definitely or probably major in computer science in college. All of the students indicated that they would major in engineering related fields in college.

The survey results were analyzed by single sample *t*-test to determine whether the observed mean is different from a set value. Results of the *t*-tests showed that students' evaluation on their computer knowledge was very positive (*mean* = 4.71, standard deviation, *S. D.* = 0.49; $t = 3.87$; $p = 0.008$). In addition, students' perceptions toward CS were positive (*mean* = 4.67, *S. D.* = 0.57; $t = 3.10$; $p = 0.02$). In terms of confidence in learning specific CS skills, such as learning to use a variety of programs, writing simple programs and explaining why a program will or will not run on a computer, students' self-evaluation was positive (*mean* = 4.63, *S. D.* = 0.57; $t = 2.68$; $p = 0.04$). Students also expressed a positive attitude toward CS (*mean* = 4.64, *S. D.* = 0.61; $t = 2.79$; $p = 0.03$). Meaning, students considered CS to mean a lot to them and perceived CS as appealing, fascinating, and interesting. However, students' interest in learning CS did not significantly decrease nor did their creativity level in learning programming. It may be due to the fact that students have already had a strong interest in learning programming before enrolling in the class so it was difficult to see a dramatic change. Since this course was the first programming experience for the majority of students, it might be difficult for students to try out new ideas or suggest new ways of performing research tasks in programming on their own (i.e., measure of *creativity*). This could be one of the reasons that *creativity* in CS did not improve after this course.

V. Conclusion

A. Course Summary

This assembly language course was developed and designed for high school students who were interested in learning programming. At the end of three or more modules, students were able to transition to different or more advanced microcontrollers or to a higher level programming language.

Six of the eight students who participated in this course work wished to continue to learn programming during the next semester. Although the students did not show a change in their interest in learning computer science, the students have a strong interest in the subject matter. Twelve new students have asked to begin learning about microcontrollers. The instructor, with full support from the administration of his high school, has begun defining a more complex embedded control programming course for the 2014-2015 school year.

B. Future Work

Ten of the new students will begin learning assembly language programming and reaping the benefit of the first semester curriculum development and adjustments. It is likely that students will be able to complete the four initial labs quickly (excluding write\read microcontroller lab) plus several others. Additional labs will be developed that may include brushless motor control, temperature measurement and display, and programming with a different microcontroller (extended instructional set or different manufacturer).

The six assembly language students plus two new students will move to a different Microchip Technology development board (PICDEM 2PLUS) and microcontroller (PIC16F1937). These students will code in the C programming language. Although this board has more internal features, the same IDE will be used and many of the PIC16F616 internal features are identical. The primary enhancement of the new board is an LCD which will permit much more complex output displays.

The six first semester students will leverage their existing knowledge of microcontroller internal functions (timers, analog-to-digital converter, instruction set) to perform much more complex labs, notably those that involve mathematics. One of their labs will be to measure voltage and current received from renewable energy projects designed by classmates and to display the power output of those designs. It should also be noted that with slight modification, the same assembly programs used in first semester could be used with the new development board and microcontroller; thus, introducing students to the important programming concept of "portability".

This work was supported in part by the Engineering Research Center Program of the National Science Foundation and the Department of Energy under NSF Award Number EEC-1041877 and the CURENT Industry Partnership Program.

Bibliography

1. Al-Dhaher, A.H.G. (2001). Integrating hardware and software for the development of microcontroller-based systems. *Microprocessors and Microsystems*, 25, 317-328.

2. Astrachan, O.L. (2000). Computer science tapestry: Exploring programming and computer science with C++, 2nd ed". Boston: McGraw-Hill.

3. Bayman, P., Mayer, R.E., & Schwartz, M.H. (1983). A diagnosis of beginning programmers' misconceptions of BASIC programming statements. *Communications of the ACM*, *26*(9), 677–679.

4. Bolanakis, D.E., Evangelakis, G.A., Glavas, E., & Kotsis, K.T. (2009). A teaching approach for bridging the gap between low-level and high-level programming using assembly language learning for small microcontrollers. *Assembly Language Learning*. 525-537.

5. Bureau of Labor Statistics, U.S. Department of Labor, *Occupational Outlook Handbook, 2012-13 Edition*, Computer and Information Research Scientists, on the Internet at http://www.bls.gov/ooh/computer-and-information-technology/computer-and-information-research-scientists.htm (visited *December 22, 2013*).

6. College Board (2012). AP Data - Archived Data 2012: National Report (12/22/2013) http://research.collegeboard.org/programs/ap/data/archived/2012

7. College Board (2012). https://apstudent.collegeboard.org/apcourse/ap-computer-science-a

8. Cuny Jan (2011). Transforming computer science education in high school. *IEEE Computer Science*, June. 107-109.

9. Djukic, D. (2011). Group projects in teaching microcontrollers. *International Journal of Electrical Engineering Education*, *48*(4), 359–371.

10. Green, P.R., Green, P.N., Bailey, M., & Foster, D.A. (2013). Design and delivery of a microcontroller engineering teaching theme. *International Journal of Electrical Engineering Education*, *50*(3), 231–238. doi:10.7227/IJEEE.50.3.3

11. Hamrita, T.K. & McClendon, R.W. (1997). A new approach for teaching microcontroller courses. *International Journal of Engineering Education*. 13(4). 269-274.

12. Microsoft Corporation (Microsoft). 2012. *A national talent strategy: Ideas for securing U.S. competitiveness and economic growth*. http://www.microsoft.com/en-us/news/download/presskits/citizenship/MSNTS.pdf

13. Puhan, J., Bürmen, Á., Tuma, T., & Fajfar, I. (2010). Teaching assembly and C language concurrently. *International Journal of Electrical Engineering Education*, *47*(2), 120–131.

14. Vahid, F. & Givargis, T. (2008). Timing is everything: Embedded systems demand early teaching of structured time-oriented programming. In Proceedings of WESE 2008: Workshop on Embedded Systems Education.1-9.

15. Wu, X., Obeng, M, & Wang, J. (2010). Project-centered pedagogy and practice in teaching microprocessor and embedded systems design to undergraduate students. IEEE. 102-105.

16. Yeong, C.F., Rahman, H.A., & Su, E.L.M. (2013). A hands-on approach to teaching microcontroller. *Journal of Education, Informatics & Cybernetics*, *11*(1), 55–59.