TEACHING CONCEPTS IN MICROCONTROLLER EDUCATION: CISC VS RICS ASSEMLBY-LEVEL PROGRAMMING

Dimosthenis E. Bolanakis
Department of Communications, Informatics and Management
Epirus Educational Institute of Technology
Greece

Konstantinos T. Kotsis
Department of Primary Education
University of Ioannina
Greece

Theodore Laopoulos
Department of Physics
Aristotle University of Thessaloniki
Greece

Abstract

This paper explores the teaching concepts in Reduced Instruction Set Computers and Complex Instruction Set Computers with reference to an assembly-level programming for small microcontroller units (MCUs). The objective of the proposed communication is to instill the confidence in the instructor regarding the selection of an effective MCU, appropriate for other than electrical/electronic engineering students.

Introduction

Microcontrollers are regularly addressed to students of the electrical/electronic engineering curriculum, but often migrate in allied disciplines so as to serve particular monitoring/control applications. Previous literature related to the important role of microcontrollers in solving engineering problems and subsequently, to the need of addressing a microcontroller-based course in various disciplines of engineering education, such as chemical engineering (Lodge, 2006), biological and agricultural engineering (Hamrita et al., 2002), and mechanical engineering (Culbreth, 2001; Giurgiutiu et al., 2004). Our lines of research are focused on the migration of microcontrollers' technology to students with software engineering orientation. Following an extensive educational research (Bolanakis et al., 2007a; Bolanakis et al., 2007b; Bolanakis et al., 2007c; Bolanakis et al., 2008a; Bolanakis et al., 2008b), an integrated microcontroller-based tutoring system is presently experienced at the Department of Communications, Informatics and Management, Arta, Greece.

Due to the fact that Microcontroller Units (MCUs) have become a popular teaching tool in various levels of engineering education, our recent propositions relied on an interdisciplinary methodology for bridging the gap between low-level and higher-level programming using assembly language learning for small microcontrollers (Bolanakis et al., 2008b). The methodology exploits the fact that freshman engineering students are regularly exposed to an introductory course on high-level programming and draws their attention to the parallelism between the assembly-level and higher-level programming topics. While this strategy acquires the composition of the fundamental high-level programming possibilities at the assembly-level using a Complex Instruction Set Computers (CISC) MCU, it is designed with a systematic attention so as it could be easily revised to any machine-depended language, using either CISC or Reduced Instruction Set Computer (RISC) MCUs. Following this particular communication, the present paper clarifies the entailed risk in addressing RISC MCUs to non electrical/ electronic engineering students. The goal of the present communication is to promote the distinguishing characteristics in the teaching of CISC and RICS assembly-level programming so as to instill the confidence in the instructor regarding the selection of an effective MCU, appropriate for other than electrical/electronic engineering students.

Criterion and Justification

Microcontroller designers are challenged to choose between two giant architectures: i.e. RISC and CISC processor units. The significant characteristic of RICS architectures, i.e. the use of simple instructions that can be executed within one clock cycle, provides large benefits in speed and minimization of the complexity measurements of algorithmic devices (El-Aawar, 2006). Accordingly, RISC architectures have become very popular tools for engineers and technicians. Alternatively, the significant characteristic of CISC architectures is related to the benefit of completing a task in as few lines of assembly code as possible. The question posed is which architecture is appropriate for educating engineering students?

An anticipated answer is that since assembly-level programming is regularly addressed for achieving the optimal performance of a low-level system implementation, students should be exposed to a RISC-based assembly language course. However, introducing students to the low-level (unstructured) programming techniques along with a reduced Instruction Set Architecture (ISA) entails the risk of causing confusion on their perception, especially when microcontrollers' technology is addressed to engineering students with insufficient background on hardware design issues (Bolanakis et al., 2007a). It is worth noting that, it is more important to provide engineering students with long-lasting

knowledge rather than one that is in line with current technological trends and might be obsolete in a few years (Hue, 2003). In consideration of an efficient microcontroller education, students should be provided with the opportunity of switching between different MCUs (Bolanakis et al., 2008b). In the remainder of the paper we reveal barriers to understanding microcontroller education, with reference to RISC-based versus CISC-based assembly-level programming that clearly defines the benefits of adopting a CISC processor unit in microcontroller education.

CISC vs RISC Assembly-level Programming

One of the main objectives of the assembly language course is to provide the learner a deeper insight into the inner mechanisms of a processor unit. However, assembly language is not considered to be a friendly teaching tool (Buckner, 2006) and therefore, the need to provide to students a clear linkage between high-level and low-level programming is often posed in the literature (Bolanakis et al., 2008b; Freudental et al., 2008; Larson & Kim, 2008). The example presented here compares the implementation of C language paradigm in RISC-like and CISC-like assembly language, and reveals students' reflections on their learning processes with reference to these kinds of architecture. The example focuses on program flow-of-control and refers to the microcontrollers M68HC908GP32 (CISC processor unit) and PIC16F84A (RISC processor unit).

Flow-of-control

One of the most important chapters in sequential programming (either high-level or low-level) is the alteration of the regular execution of a program. Students are primarily taught how to provide a choice of action within programs. Flow-of-control statements in high-level programming use a test expression that regularly admits relational operators. According to the results of the evaluated expression, the body of the statement is either executed or aborted. On the other hand, low-level actions in assembly language evaluate flags of a status register embedded in the central processing unit (CPU) so as to determine a conditional branch to an effective address. Due to the necessary actions related to the CPU manipulation (which are regularly missing in high-level programming) students are exposed to an onslaught of new information early in their assembly language education.

Table 1: Relational Operators and CISC Assembly Instructions

Relational operators		Mnemonics	
<	Less than	BLO	Branch if lower
<=	Less than or equal to	BLS	Branch if lower or same
>	Greater than	BHI	Branch if higher
>=	Greater than or equal to	BHS	Branch if higher or same
==	Equal to	BEQ	Branch if equal
!=	Not equal to	BNE	Branch if not equal

CISC microcontrollers employ a set of assembly language instructions (mnemonics) that could be easily associated with relational operators. Compared to the ISA of a RISC MCU, this option provides the benefit of minimizing information during the initial learning stages for students. While these mnemonics evaluate the CPU status (flag) register, their description permits omission of this particular information and thus, facilitates the passage from a higher to a lower level of programming. In addition, CISC microcontrollers encompass mnemonics that perform subtraction between two registers without changing their content. Those mnemonics provide significant advantages during the assembly code development process. Tables 1 and 2 present relational-like and compare mnemonics respectively, which are referred to the microcontroller MC68HC908GP32. In the following we give an *if* statement example that is addressed to reveal the benefits in CISC-like assembly-level programming with reference to students' reflections on their learning processes.

Table 2: Compare Instructions in CISC Assembly Language

Compare mnemonics	
CMPA	Compare A (accumulator) with memory
CMPX	Compare X (index low) with memory
CBEQA	Compare accumulator with immediate and branch if equal
CBEQX	Compare X (index low) with immediate and branch if equal

IF example: Figure 1 a) presents an *if* example in C, while Figure 1 b) and c) presents the equivalent code in MC68HC908GP32 and PIC16F84A assembly language respectively.

Figure 1: C and Assembly Code (*if* example*)*

```
1  if (x>y & x>3)
2     {
3      "statements"
4     }
```

a)

```
1  if       lda   x        ;a <= x
2           cmpa  y        ;a-y (i.e x-y)
3           bls   if_end   ;skip if_body if x<=y

4           cmpa  #3       ;a-3 (i.e x-3)
5           bls   if_end   ;skip if_body if x<=3

6  if_body  "statements"

7  if_end
```

b)

```
1  if       movf  x,0       ;w <= x
2           subwf y,0       ;w <= w-y (i.e. w <= x-y)
3           btfsc status,C  ;test carry flag and skip next
                            ;instruction if C=0 (i.e. x>y)
4           goto  if_end    ;skip if_body if x<y

5           btfsc status,Z  ;test zero flag and skip next
                            ;instruction if Z=0 (i.e. x!=y)
6           goto  if_end    ;skip if_body if x==y

7           movf  x,0       ;w <= x
8           subwf 3h,0      ;w <= w-3 (i.e w <= x-3)
9           btfsc status,C  ;test carry flag and skip next
                            ;instruction if C=0 (i.e. x>3)
10          goto  if_end    ;skip if_body if x<3

11          btfsc status,Z  ;test zero flag and skip next
                            ;instruction if Z=0 (i.e. X!=3)
12          goto  if_end    ;skip if_body if x==3

13 if_body  "statements"

14 if_end
```

c)

Regarding the CISC-based assembly code in Figure 1 b), students are primarily introduced to the relational-like mnemonics (Table 1) and thereafter, it is explained to them that test expressions in the assembly-level are formed using a combination of compare (Table 2) and relational-like mnemonics. Compare mnemonics are used to prepare a condition, while the subsequent relational-like mnemonics are used to verify the condition. If the evaluated condition is found to be true, relational-like mnemonics alter the program flow to the memory location defined by a descriptive label. Otherwise, program flow continues to the subsequent instruction in the assembly code. Due to the fact that compare instructions are performed through the general purpose registers of the CPU, i.e. accumulator (A) and index register low (X), the content of the tested variable should be previously assigned to either A or X register. Another point that should be taken into account is the selection of the relational-like mnemonic that is associated to the opposite high-level operator, so as to ease the assembly code development process. Specifically, it is more convenient to evaluate the false condition and abort the execution of the flow-of-control statement rather than evaluate the true condition and fetch the body of the clause, in case the evaluated condition is verified. Accordingly, lines 1–3 in Figure 1 b) evaluate the condition $x \leq y$ end abort the execution of the statement in case it is found to be true, while lines 4–5 perform the same action in case the condition $x \leq 3$ is confirmed. At this point it is worth noting that, while compare mnemonics affect the flag register and relational-like instructions check the status of the register bits, that type of information could be easily omitted during the early lessons of assembly language learning. Therefore, students can focus on the necessary actions for switching from high-level statements to assembly-level techniques.

Contrary to CISC-like assembly language programming, this kind of information is necessary in RISC-like programming. Considering the RISC-based assembly code in Figure 1b), students should be primarily introduced to the CPU flag register and thereafter, learn which particular mnemonics have an effect on flag register and in which register bits. In addition, during the code development process, the students deal with the situation of thinking how the result of an arithmetic operation affects the flag register and subsequently the program flow. The lack of compare mnemonics present students with the responsibility of reassigning general purpose registers to the content of the tested variable before every arithmetic operation. Accordingly, line 1 assigns variable x to the CPU general purpose register (w), line 2 subtracts y variable from w and line 3 tests carry flag (C) in status register. If C = 1 (i.e. x < y) the subsequent instruction (line 4) is executed and thus, the body of the *if* clause is aborted. Otherwise, line 5 evaluates the zero flag (Z) and proceeds to the same action in case Z = 1 (i.e. x = y). Thus, if Z = 1 line 6 is executed and the body of the *if* clause is aborted. The same action is repeated in lines 7–12 so as to determine the execution of the *if* body in case x>3.

Assessment

In an effort to assess the value of the proposed CISC-based pedagogy, an anonymous questionnaire was administered to 39 students at the Department of Communications, Informatics and Management, Epirus Educational Institute of Technology, Arta, Greece. Table 3 summarizes the results of the assessment survey. The first two questions explore the value of the proposed methodology in obtaining the educational benefits of the assembly language programming course, that is, the understanding of computer architecture and the improvement of high-level programming skills. The third question examines the value of the proposed pedagogy in facilitating learning.

Positive results of the first two questions find the students agree with the effectiveness of the proposed methodology to their education. The expected higher score in the first question confirms that the assembly language learning helps more in understanding how a computer machine works and less in improving high-level programming skills. The high score of the third question proves that the proposed CISC-based approach facilitates students' learning. The fact that none of the students believes that the current method helped in the conception of the low-level programming issues, shortly or at all, highlights the educational benefits of a CISC-based approach on students' perception.

Table 3: Assessment Survey

No	Questions	Very much (5)	Much (4)	Enough (3)	Shortly (2)	At all (1)	average
1	Do you believe that the code development in low-level languages supports the understanding of the internal structure and the way a computer machine works?	3	14	17	4	1	3,36
2	Do you believe that the understanding of the code construction at the machine level can prove your skills on high-level programming?	1	13	17	7	1	3,15
3	Do you believe that the assembly language tutoring using structured peudocode of a familiar high-level language as an interim step facilitates the conception of the low-level programming issues?	14	16	9	0	0	4,13

Discussion and Concluding Remarks

The lack of structures in the assembly language requires a significant effort for transcending the limits between high-level and low-level programming. Therefore, our recent propositions relied on an interdisciplinary methodology for bridging the gap between low-level and higher-level programming using assembly language learning for small microcontrollers (Bolanakis et al., 2008b), while the present paper clarifies the reason for adopting a CISC CPU in microcontroller education. Because CISC microcontrollers employ a set of mnemonics that could be easily associated with relational operators, as well as compare mnemonics that create results on the fly, their usage permits minimization of information during the initial learning stages. This provides students with the advantage of focusing on low-level programming techniques for converting high-level programming constructs at the machine level; rather than focusing on too much technical detail regarding the effect of particular mnemonics on the CPU flag register. It is our belief the latter information should be provided to the students after their familiarization with the low-level programming techniques; otherwise students might face confusion and disorientation from the main purpose of the course. Following the proposed educational sequence, students could easily exploit low-level programming techniques in the subsequent advanced lessons of interrupts, timers, serial communication interfaces, etc. Due to the fact that RISC architectures have become very popular tools for engineers and technicians, the issues discussed within this paper reveal the entailed risk in addressing these

architectures to students with insufficient background on topics related to hardware domain. The purpose of this communication is to instill the confidence in the instructors regarding the selection of an effective MCU, appropriate for other than electrical/electronic engineering students.

References

Bolanakis, D. E., Evangelakis, G. A., Glavas, E., & Kotsis, K. T. (2008a). Teaching the addressing modes of the M68HC08 CPU by means of a practicable lesson. *The 11th IASTED International Conference on Computers and Advance Technology in Education* (pp. 446–450). Crete, Greece.

Bolanakis, D. E., Evangelakis, G. A., Glavas, E., & Kotsis, K. T. (2008b). A teaching approach for bridging the gap between low-level and high-level programming using assembly language learning for small microcontrollers. *Computer Applications in Engineering Education*, published online on April, 16, 2009 (early view in advance of print).

Bolanakis, D. E., Glavas, E, & Evangelakis, G. A. (2007a). An integrated microcontroller-based tutoring system for computer architecture laboratory course. *International Journal of Engineering Education*, *26*(4), 785–798.

Bolanakis, D. E., Glavas, E, & Evangelakis, G. A. (2007b). Levin's approach for microcontrollers tutoring. *6th ASEE Global Colloquium on Engineering Education* (pp. 1–11). Istanbul, Turkey.

Bolanakis, D. E., Glavas, E, & Evangelakis, G. A. (2007c). A multidisciplinary educational board system for microcontrollers: Considerations in design for technically accurate custom-made platforms. *The 1st International Symposium on Information Technologies and Applications in Education* (pp. 391–395). Kunming, P. R. China.

Buckner, K. (2006). A non traditional approach to an assembly language course. *Journal of Computing Sciences in Colleagues*, *22*(1), 179–186.

Culbreth, W. (2001). Meeting the needs of industry: Development of a microcontroller course for mechanical engineers. *ASEE Annual Conference & Exposition* (pp. 1–9). Albuquerque, New Mexico.

El-Aawar, H. (2006). *CISC vs RISC hardware and programming complexity measures of addressing modes.* International Conference on Perspective Technologies and Methods in MEMS Design (MEMSTECH). Lviv-Polyana, Ukraine.

Freudental, E. A., Carter, B. A., Kautz, F. F., & Ogrey A. N. (2008). Work in progress–combined introduction of C and assembly with a focus on reduction of high-level language constructs. *ASEE/IEEE Frontiers in Education Conference* (pp. S2H/3–S2H/4). Saratoga Springs, NY.

Giurgiutiu, V., Lyons, J., & Rocheleau D. (2004). Mechatronics/microcontrollers education for mechanical engineering students at the University of South Carolina. *ASEE Annual Conference & Exposition* (pp. 1–10). Salt Lake City, Utah.

Hamrita, T. K. (2002). Micro-controllers in the biological and agricultural engineering curriculum at the University of Georgia. *ASEE Annual Conference & Exposition* (pp. 1–6). Montreal, Quebec, Canada.

Hu, S. C. (2003). A wholesome ECE education. *IEEE Transactions on Education*, *46*(4), 444–451.

Larson, E., & Kim, M. O. (2008). A simple but realistic assembly language for a course in computer organization. *ASEE/IEEE Frontiers in Education Conference* (pp. S2H/3–S2H/4). Saratoga Springs, NY.

Lodge, K. (2006). The programming of a micro-controller as the laboratory component in process control for undergraduates in chemical engineering. *ASEE Annual Conference & Exposition* (pp. 1–12). Chicago, Illinois.