

Work in Progress - Combined Introduction of C and Assembly with a Focus on Reduction of High-level Language Constructs

Eric A. Freudenthal, Brian A. Carter, Frederick F. Kautz, and Alexandria N. Ogrey
University of Texas at El Paso, efreudenthal@utep.edu, {bacarter,ffkautz,anogrey}@miners.utep.edu

Abstract - This paper describes the reform of a sophomore-level course in computer organization for the Computer Science BS curriculum at the University of Texas at El Paso, an urban minority-serving institution, where Java and integrated IDEs have been adopted as the only language and development environments used in the first three semesters of study. This effort was motivated by faculty observations and industry feedback indicating that upper-division students and graduates were failing to achieve mastery of non-garbage-collected, strictly imperative languages, such as C. The similarity of C variable semantics to the underlying machine model enables simultaneous mastery of both C and assembly-language programming and exposes implementation details that are difficult to teach independently, such as subroutine linkage and management of stack frame. An online lab manual has been developed for this course that is freely available for extension or use by other institutions. In this paper, we report on pedagogical techniques for facilitating student understanding of the relationships between high-level language constructs, such as algebraic expression syntax, block-structured control-flow structures, and composite data types, and their implementations in machine code.

Index Terms - computer architecture, CSE, assembly language, compilation

INTRODUCTION

Until the mid-1990's, the introductory sequence of many Computer Science curricula included two courses that introduced students to, programming, and common data structures using one of a variety of procedural languages that directly exposed students to management of memory and the use of pointers. These concepts are necessary for understanding the relationship between high level languages, run-time environments, and underlying architecture. Understandings and skills at this level are common with EE curriculum still in use.

In response to the wide adoption of object-oriented languages with "reference" abstractions and automatic garbage collection such as Java that reduce the difficulty of engineering and debugging complex systems, a now-popular "objects first" curriculum design was introduced and is now

listed by the ACM as an acceptable implementation strategy Computer Science education.

This wide adoption of object oriented languages and their seamlessly integrated IDEs has enabled lower division students to construct more complex programs at the cost of decreasing their understanding of memory management and system-level tools such as compilers and linkers. These understandings are critical for developers of operating systems and embedded controllers. This learning deficit has been observed by industry and faculty at UTEP and other institutions that adopted objects-first curricula. [1]

This paper describes an experimental reform to a 4th semester course whose principal learning outcome is an understanding of computer system organization and assembly language. In the reformed course, the C programming language, which is commonly used for low-level system implementation, bridges between the familiar *syntax* of Java and the *semantics* of the underlying system.

The pre-reform computer organization course focused on foundational concepts such as machine instructions, registers, the random-access memory model, and a generalized fetch-execute cycle [2]. Projects included assembly-language programming of a Motorola M68HC11 processor embedded within a robot. The reformed course [3], which uses a different embedded target, integrates the study of C and is thus also able to focus on the implementation of high-level language features and the linkage between C and assembly-language routines. Student labs use traditional command-line tools including bash, gcc, as, ld, and make.

The reformed course's outcomes are a superset of the original, with extensions including (1) understanding of C and its runtime environment, (2) parse trees, (3) implementation of simple dynamic memory management, and (4) usage of traditional command-line development tools. While no formal evaluation has yet been conducted, examinations and projects indicate that most students develop competency in the both the previous and extended outcomes, and students have expressed enthusiasm for learning C due to its high perceived commercial relevance.

Ironically, this work was inspired by the recent work of Yale Patt who developed a architecture-first (a.k.a. "breadth-first") curricula that introduces students to underlying architecture and machine language concepts prior to high-level language programming in C [4], [5]. The approach

described here is complementary to Patt's since it exploits understanding gained from the prior study of high-level (and even object-oriented) languages to facilitate the combined understanding of C and computer organization.

ADOPTION OF THE TI MSP430 MICROCONTROLLER

In the spring of 2008, the TI MSP430 embedded controller was adopted. Like the HC11, the MPS430 is supported by the free GNU development tools. Advantages of this controller over the previously used M68HC11 include a smaller orthogonal (and thus easier to learn) instruction set and inexpensive (~\$20) development kits.

DECONSTRUCTING HIGH-LEVEL PROGRAMMING CONSTRUCTS INTO MACHINE-LEVEL IMPLEMENTATIONS

Assembly language programming is taught through examination of strategies for translating high level C-language constructs into assembly language. This is accomplished through the discussion and manual application of techniques that generally are reserved for courses on compiler construction. As illustrated in Figure 1, manually generated parse trees are introduced as a technique for identifying sub-expressions, determining evaluation order, and managing temporary variables in a manner that is easily understood and applied by students.

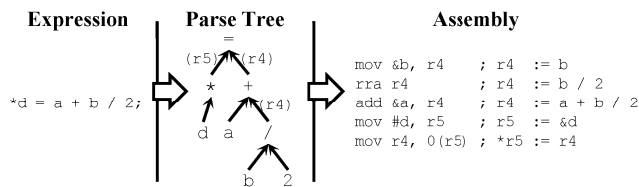


FIGURE 1

ILLUSTRATION OF A PARSE TREE USED AS AN AID TO CONVERT A C-STYLE ARITHMETIC EXPRESSION TO ASSEMBLY-LANGUAGE CODE.

CONTROL FLOW

Students who are only familiar with block-structured programming can easily be confused by the translation of block-structured control-flow constructs (e.g *if-then-else* clauses, *switch* statements and *and for* or *while*-loops) to equivalent branching assembly-language code. Branching instructions have similar semantics to the C (and FORTRAN) goto statement. Rather than introducing translation templates, as fixed idioms, the course exploits C's support for both (conditional) goto and block-structured primitives to enable the students to explore the logical reduction of block-structured programs to "goto C." As is illustrated by Figure 2, the subsequent translation of goto C to (MSP430) assembly language is straightforward.

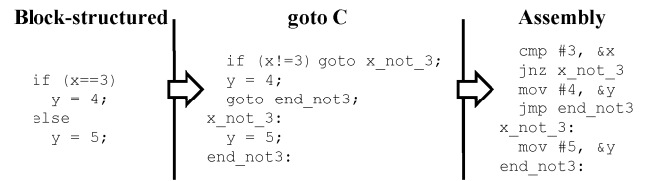


FIGURE 2

REDUCTION OF BLOCK-STRUCTURED C TO BRANCHING CODE

COMPOSITE DATA TYPES

The course also includes labs in which students manipulate composite data types declared using C's struct primitive. Lectures and exercises examine the memory representation of abstract data types such as trees and linked-lists whose members are defined as structs. A linked list example is illustrated in Figure 3. Lab projects include programs that manipulate abstract data types (such as linked lists) oth accessed by mainline code in C and by I/O handlers written in assembly language. This figure also illustrates a technique used by two-pass assemblers to determine the address of instructions prior to opcode generation.

REFERENCES

- [1] Dewar, Robert and Sconberg, Edmond, "Computer Science Education: Where are the Software Engineers of Tomorrow," *STSC Crosstalk*, January 2008.
- [2] Teller, Patricia; Nieto, Manuel; and Roach, Steve, "Combining learning strategies and tools in a first course in computer architecture," *IEEE proc. 2003 Workshop on Computer Architecture Education*.
- [3] Freudenthal, E., Carter, B., Kautz, F., Ogrey, A., Preston, R., Walton, A., "Integration of C into an Introductory Course in Computer Organization," *ASEE 2007 ECE Division Program*.
- [4] Patt, Yale, "Education in Computer Science and Computer Engineering Starts with Computer Architecture," *ACM 1996 proc. 1996 Workshop on Computer Architecture Education*.
- [5] Patt, Yale and Patel, Sanjay, *Introduction to Computing Systems*. McGraw Hill, 2004. ISBN 0-07-121503-4.
- [6] Association for Computing Machinery, *Computing Curricula 2001 Computer Science*, ACM, <http://www.sigcse.org/cc2001/cc2001.pdf>

AUTHOR INFORMATION

Eric A. Freudenthal is a member of the Computer Science Faculty at the University of Texas at El Paso.

Brian A. Carter, Frederick F. Kautz, and Alexandria N. Ogrey are undergraduate students studying Computer Science at the University of Texas at El Paso.

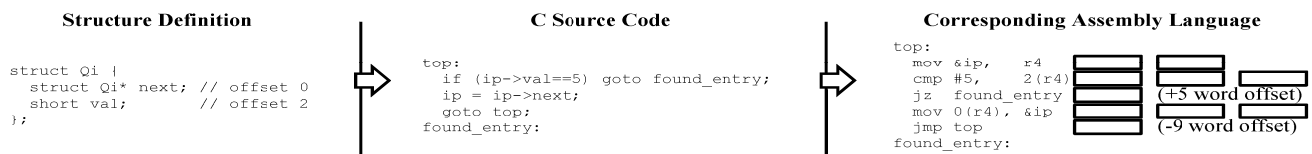


FIGURE 3

COMPOSITE DATA TYPE EXAMPLE AND ILLUSTRATION OF MEMORY ALLOCATION FOR INSTRUCTION WORDS.