Teaching Students Java Bytecode Using Lego Mindstorms Robots

Michael J Jipping
Department of Computer Science
Hope College
Holland, MI 49423
jipping@cs.hope.edu

Cameron Calka
Department of Computer Science
Hope College
Holland, MI 49423
cameron.calka@hope.edu

Brian O’Neill
Department of Mathematics and Computer Science
Saint Joseph's University
Philadelphia, PA 19131
brian.oneill2@sju.edu

Christopher R. Padilla
Department of Computer Science
University of Hartford
West Hartford, CT 06117
padilla@hartford.edu

ABSTRACT
Assembly language is a valuable subject to teach, yet one of the most underappreciated by students. Students do not see the need for assembly language and they get through it as quickly as possible. Given the time in a computer organization course to teach assembly language, assignments – either quick-and-easy ones or those contrived-to-teach-a-point – contribute to this attitude of malaise. This paper documents a project to inject some fun into the teaching of assembly language. First, we switch the language to Java bytecode. Second, we use unique assignments to use Java bytecode with Lego Mindstorms robotics.

Categories and Subject Descriptors
K.3 [Computers & Education]: Computer and Information Science Education – Computer Science Education.

General Terms
Experimentation, Design, Human Factors

Keywords
Assembly language, computer organization, robotics, Lego Mindstorms

1 INTRODUCTION
In the computer science curriculum, the Computer Organization course stands out because of the time frame in which it is taught and the sheer variety of topics which must covered. The course is introductory by nature and needs to be approachable by first-year students. It combines discrete mathematics with hardware design and assembly language programming. It requires that new and difficult subject matter be delivered to the student through hands-on experimentation-based activities. Active learning and experimentation in a Computer Organization course can be difficult because of the variety of material in the course and the speed with which it is delivered.

In addition, the course focuses on aspects of computer science that are not covered in many other places. For example, there are not many other classes that cover assembly language programming...or use it, for that matter.

Assembly language programming can become a part of the Computer Organization course that is simply tolerated. There is rarely enough time to teach a complete assembly language programming section (much less a class) and assignments can be designed to give students an overview only – ones that involve either quick, survey-oriented assignments or those contrived to point out specific aspects of assembly language. In an experimentation-based course, assembly language can be dry and experience-free task.

This paper documents an approach that we have designed to tackle these issues in teaching assembly language programming. We choose to teach Java bytecode, for reasons we will spell out, and we teach Java bytecode using Lego Mindstorms robotics kits. We begin by giving the motivation for choosing bytecode. We then describe the system used to get Lego robots to obey Java programming. We conclude the paper by describing our system of assignments that get students working.

2 BACKGROUND AND MOTIVATION
The Computer Organization course is often a course with several separate components. The course is meant to allow students to apply discrete mathematics concepts to digital logic
design, to briefly explore computer architecture, and to dabble in assembly language programming, often as a way to exemplify the circuit and architecture material. Along the way, students are asked to experiment in all of these areas. It is a fast-paced course with little time to spend learning development systems.

Time constraints make the teaching of assembly language, which is an important part of the Computer Organization course, quite difficult. It is, in fact, a complete programming language in itself and teaching a programming language – one might devote an entire course to doing this – in a brief section of a larger course is not easy. Making assignments in this context is very challenging: it is difficult to assign anything more than simple assignments contrived to demonstrate specific points.

The importance of assembly language is in two areas. First, it is a language that supports other languages and, in this role, it helps in the understanding of higher-level programming. The complexities of algorithms and data structures stand out when you see them in an assembly language. Second, assembly language programming directly addresses the machine architecture on which it executes and, in this role, it helps understanding and appreciation of that architecture. There are many “head knowledge” concepts that just do not hit home until you have to program using them. The necessity of indirect addressing with a load/store RISC architecture suddenly makes sense when you must load variables from memory in a SPARC architecture assembly language program.

Our Computer Organization course has faced all the issues discussed above: tight time constraints, the need to fit a lot of subject matter into this one course, and treatment of topics as introductions. In addition, we recognized the value of assembly language programming, but taught our assembly language section with contrived examples, designed to show off language features but divorced from a student’s experience with programming. We wanted to reemphasize the importance of assembly language programming by playing to a student’s frame of reference and by placing programming in a context where a student might enjoy programming (and even get inspired).

We started by changing the language taught from SPARC assembler to Java bytecode. We did this for several reasons. First, our students’ experiences are more likely to include Java and the Java virtual machine than C++ on a Windows or Unix system. One of the ways we use to illustrate assembly language is to describe programs in the context of higher level languages. Students have programmed in Java by the time they take this course and examples in Java make more sense to them. A second reason we changed assembly languages is the opportunity to discuss “interesting” machine architectures. Java bytecode allows us to discuss the architecture of the Java Virtual Machine (JVM). This discussion allows us to apply many architecture design issues in a new way. Issues involving stack-based architectures, instruction set design, and machine performance can be applied to the Java Virtual Machine. Third, the tools available – from disassemblers to profiling tools – are more plentiful with Java and they are free. Finally, we chose Java bytecode because the programs we can write are more unique and interesting. For example, we could not have applied SPARC assembly language to Lego Mindstorms.

Java bytecode is known as the assembly language in Java. It is used as a middle ground between the Java programming language and the Java Virtual Machine. Bytecode in Java plays the same role as assembly language for hardware CPUs, yet there some important differences. It has no mnemonic definition; the official specification for bytecode describes each instruction and its effect on the JVM, but does not give mnemonics. This means that, while official compilers exist that generate bytecode, there is not official assembler.

In fact, there are many different Java bytecode assemblers available due to this lack of an official assembler. For our research we opted to use the Jasmin assembler [3]. Jasmin was originally written as a companion tool to the book Java Virtual Machine [2] and is now a SourceForge open source project. Jasmin is similar to other assemblers. The classes are first written in Jasmin and then are converted to binary Java class files which can then be loaded and executed by any Java runtime system.

3 JAVA BYTECODE AND MINDSTORMS

For our research we chose to use the Lego Mindstorms Robotics Invention System (RIS) 2.0. This kit includes 718 Lego pieces, one Infrared USB Tower, one RCX brick, one light sensor, and two touch sensors. The Infrared USB Tower is connected to the PC and is used to transmit data, such as the firmware and programs, to the Lego Mindstorms RCX brick. The RCX, Robotic Command eXplorer, brick is the main component of the Lego Mindstorms kit. Measuring at approximately 6 cm x 10 cm x 4 cm, the RCX brick is a very small and powerful device. There is a Central Processing Unit, CPU, inside of the RCX which is a Hitachi H8/3292 series microcontroller. The CPU operates at 16 MHz which is more than enough processing power to operate the sensors and motors that can be attached to the brick. The RCX also controls 32 kB of external RAM which is used to store both the firmware and the user’s programs. In order to communicate and upload the firmware and user programs, the RCX contains an Infrared device that is located in the front of the device. A light sensor can be used to read the values of different light input and allow the robot to react accordingly. A touch sensor allows the robot to react to three different tactile states. These include whether the touch sensor is currently pushed, is currently not pushed and has been pushed and released.

The use of Java bytecode with Lego Mindstorms requires loading a different firmware onto the RCX brick. We chose to use the Java environment known as leJOS which is directly compatible with RIS system. LeJOS is a system that includes JVM firmware for the Lego Mindstorms and tools to convert and upload Java class files to the RCX brick. It was originally developed from another Lego Mindstorm firmware available known as TinyVM. This SourceForge project [4] was developed in 1999 and eventually matured to what is known as leJOS today.

---

1 At the time of this writing, the next version of Mindstorms, known as NXT, is due anytime. We still are going with the older RIS system because (a) it is readily available (while NXT is not), and (b) the Java Virtual Machine system we tested is available and works well on the older system.
Normally, leJOS works with Java files and converts the Java files to class files and uploads the files to the RCX brick. Instead of having leJOS convert Java files to class files, however, we take Jasmin files, convert them to class files with Jasmin, and upload these files to the RCX brick via leJOS. In order to speed this process up we created a batch file which will take all the Jasmin (.j) files and convert them to class files using Jasmin. Once this is complete, the batch file will upload the class files to the RCX brick via leJOS. This batch file assumes that all the Jasmin files compile correctly.

In order to limit the amount of mistakes made by the student we also created a defined language plug-in for a text editor (Notepad++ [1]), allowing the students to get a visualization of the different components inside of a Jasmin file. This defined language splits Jasmin keywords into color groups and displays them accordingly. Using this plug-in allows the student to view the code as if they were writing the code in an IDE. While Notepad++ lacks error detections, the defined language plug-in helps to alleviate common spelling mistakes of keywords.

There are other tools are useful as students design code for the leJOS environment. We found that when creating a program it proves to be beneficial for the programmer first create the program in a familiar language, such as Java, and then continue to program in Jasmin. Doing so will allow the programmer a better understanding of what they are attempting to program in a Jasmin. We chose to use the Eclipse IDE as our programming environment when coding in Java. This allowed us to use a leJOS plug in that is available that allows the user to compile and upload Java files through Eclipse rather then using command prompt. This helped to speed up the process of our preliminary steps of programming in Jasmin.

4 ASSIGNMENTS

In our choice of leJOS, we have a robotic control system that was programmable in Java. This choice is supported by development tools. We next needed a set of class assignments that would illustrate Java bytecode, but not in a contrived way.

Our assignments are designed to have the students write and code an algorithm for a maze-solving robot. To achieve this, we designed our own algorithms, maze, class structure, and assignments using leJOS and Jasmin. The students will start with our class structure and files. Over three assignments, the students will replace our code with their own to introduce the design of their robots and algorithms. After each assignment, the students will download the code to the robot and place it into the maze.

4.1 Design of the Maze

The floor of the maze is made of foam posterboard, while the walls are built out of Duplos, the Lego blocks intended for preschool-aged children. We chose these blocks because of their size and low price. The walls are approximately 6 Duplos high and 1 Duplo thick. The maze is built on a grid design, with each grid square having a length 14 Duplo pegs. Our example maze is a 4x4 grid, however any grid alignment should work; it need not be rectangular. In order to stabilize the maze, we used a “bridge” made of Duplos to connect separate wall sections.

This prevents the robots from bending the maze by pushing against the wall. Because of the height of the bridge, the robots may not be any taller than the walls.

The red line running down the center of each alley is made from electrical tape. At the intersections in the maze, the red tape is overlaid with a black rectangle, also made from electrical tape. Students can use light sensors to follow the line, using the black sections to know that a decision needs to be made.

The maze was designed so that the start and finish are both along the outside wall. The start is closed off, so that there is no chance of the robot exiting the maze through the entrance. If a robot somehow returned to the start, it would only notice a dead-end.

This type of maze design allows several different solutions. We provide two of them, but there are others. One possible solution uses a light sensor programmed to act as a proximity sensor. It navigates the maze by following the red line with one light sensor, and using the converted light sensor to avoid oncoming walls. Another design follows the walls in the maze using a light sensor pointed to the side and a touch sensor in front.

4.2 Context of the Assignments

By the time the students are given the first assignment, it is expected that they will already have received some instruction in Java bytecode. We will give the students a guide to bytecode and Jasmin formatting. They will also get a summary of important functions from the leJOS API, and tips for debugging and running their programs. In addition, the students will see a “HelloWorld” program on the RCX performed and be given the code in Java and bytecode. This will give them some points of reference as they try to write their own bytecode. These experiences are done through in-class lab experiments.

Prior to considering their own algorithms for their robot, the students will see two successful designs in action. The first algorithm, the BumpBot, uses a touch sensor to follow the right-hand wall until it reaches the exit. The second algorithm, the Line Follower, uses a red line down the center of each alley to make its way through the maze. A black box denotes an intersection in the maze where the robot must make a decision about which way to go. In this case, the algorithm tells the robot to turn right. If the robot hits a wall, it turns around.
Students receive copies of our class files. To pilot the robot through the maze, they will package what code they have written with any code they need of ours. However, to dissuade students from using a decompiler to write their bytecode, the class files we give them will have been run through an obfuscator to scramble our code.

### 4.3 Assignments

Using the class structure shown in Figure 2, the students will replace our class files with their own over three separate assignments.

In the first assignment, students will remove and replace the TouchSensor and LightSensor classes. Students will be responsible for both classes regardless of whether they plan to use each type of sensor. In the assignment description, we will delineate any methods that are necessary for this class to work with the other classes. Students will also be encouraged to add any methods that they feel will be necessary for algorithms they may be considering. The students will be given approximately 2-3 days to complete the assignment. The robot will be placed in the maze to test the LightSensor and TouchSensor code.

For the LightSensor class, we expect the following functions:

- void activate()
- void addSensorListener(SensorListener)
- boolean betweenValues(int, int)
- int getId()
- int getValue()
- boolean isDark()
- boolean isLight()
- void off()

Each of these functions are described in the BULB API, which will be provided to students.

For the TouchSensor class, we expect a similar list of functions:

- void activate()
- void addSensorListener(SensorListener)
- int getId()
- boolean isPressed()
- void off()

For the second assignment, students will replace our Motors class with their own. The Motors class governs the movement of the robot. As with the previous assignment, the students will be told what methods are necessary for the robot to behave properly. Again, the students will be encouraged to add other methods to work with their own designs. The students will be given one week to complete this assignment. Upon completion, the robot will be tested in the maze.
As with the sensor classes, we expect certain methods to be present in student versions of the Motors class:

- void brake()
- void coast()
- void forward()
- void forward(long)
- void fwdDist(int)
- void spin(float)
- void spinLeft(double)
- void spinRight(double)
- void turnLeft()
- void turnLeft(long)
- void turnRight()
- void turnRight(long)

For the final assignment, the students will replace the MazeSolver class that governs the actions of the robot. This class interacts with the Behavior interface, which is part of the leJOS API, and can be used with an Arbitrator object to prioritize the different actions of a robot. This is where the design of the maze-solving algorithms will become evident. Students may either attempt to duplicate the behaviors they saw in the example robots, or create their own with a new algorithm. Alternately, if the students want to keep working with one of the provided designs rather than trying to implement their own, they simply need to replace the MazeSolver class. The students will have a week and a half to work on this assignment, although it is expected that they have been considering their algorithms throughout the unit. As with each assignment, the students will place their robot in the maze to test its performance.

5 Conclusion

This paper has presented the reasons and mechanics behind using Java bytecode and Lego Mindstorms Robots to teach assembly language in the Computer Organization course. We described why we chose Java bytecode over other assembly languages. We described the methods behind putting Java bytecode on Mindstorms Robots. We also created assignments for how it would be possible to use Lego Mindstorms and Java bytecode in a classroom environment. These assignments will be used in classes at Hope College.

Currently, two additional items were created to help students with Java bytecode. Future expansions upon this may include a plug-in for the popular Eclipse IDE, which would allow students to upload files directly from the IDE onto the Mindstorms RCX. The ability to add syntax checking for Java bytecode would also encourage more use of Java bytecode and these assignments.

Acknowledgement

The work described in this paper was performed under a CCLI program award from the National Science Foundation, award number DUE-0310757.

References