A Microprocessor Trainer Simulator

William P. Lovegrove
Bob Jones University
Greenville, SC 29614

Abstract

Historically, a two semester sequence of courses in microprocessor software and interfacing was taught using a processor such as the Motorola 6800. However, advances in the field have necessitated teaching more advanced microprocessors (such as the Intel 80x86 family) in the software course.

For several reasons we continue to teach the interfacing course using the Motorola 6800. Reasons include legacy laboratory hardware, tractability of the 8 bit hardware bus, and the popularity of the 6800 family derivatives as embedded controllers. Students are now required to master the 6800 architecture and assembly language in a short time at the beginning of this course.

To ease this learning process, we have developed a software simulation of the popular Heathkit Model ET-3400A microprocessor trainer. All hardware experiments are carried out on the physical trainers, but students can do software development at home without physically possessing a trainer. Anecdotal evidence to date suggests that the simulator is of significant benefit to the students in quickly mastering the 6800 architecture.

The Old Curriculum

In the 1970's and 1980's, when microprocessors were in their infancy, many universities taught a two or three semester sequence of classes centered on one of the popular eight bit processors, typically the Motorola 6800 or the Intel 8085. Typically one semester was devoted to learning the processor architecture and instruction set. A second semester was given to hardware interfacing.

As technology progressed and microprocessors advanced, these early eight bit processors quickly became obsolete. Their use in university curriculums continued, for several reasons. First, the investment in equipment and curriculum development required to change processors was substantial. Second, their simple architectures made them tractable within the constraints of a single semester. Finally, they continued to see wide use as processors in embedded systems, a factor that continues to this day.

The Move to 80x86 Software

Several recent developments have led to a re-examination of this philosophy. Not too long ago, the majority of embedded system programming was done in assembly language. Now it is nearly all done in C or some other higher level programming language. Only the smallest and simplest systems are generally programmed entirely in assembly language. Of course, system software will continue to be written in assembly language. That, however, is not the application in mind in most engineering departments.

In addition, most modern processors contain a growing list of architectural features that are not easily addressed within the bounds of the older 8-bit processors. Capabilities such as floating point arithmetic, cache memory, pipelining, and high level language support increasingly need to be addressed.

Furthermore, the wealth of high quality inexpensive software and hardware tools available on the IBM-PC compatible platform have made the 80x86 architecture far more tractable within a semester.

Finally, and most persuasively for us, the 80x86 is beginning to see widespread use in embedded systems. The IBM-PC compatible architecture is now so ubiquitous than numerous single-board or even single-chip implementations are available as a drop-in single component.

For all of these reasons, Bob Jones University and many other engineering schools have abandoned the older 8-bit architectures in favor of the 80x86 architecture. Our first microprocessor course is now an 80x86 architecture and programming course.

By 80x86, we refer to the entire family of subsequent processors that have come from the original Intel 8086. We concentrate on the core 8086 instruction set, and give brief coverage to the added features particularly in the 80386 and 80486 processors.

6800 Hardware

Many schools have also migrated to the IBM-PC as a platform on which to study microprocessor hardware and interfacing. We have chosen not to do so, for several reasons. First is the availability of existing 6800 hardware. This includes not only the 6800 trainers that are the focus of this paper, but also in-circuit emulators and logic analyzer probes.

Second, the 6800 and its family derivatives continue to see widespread use as embedded processors. The 68HC11 in particular is a popular processor. Our embedded systems course is based on the 68HC11 architecture.

Third, switching to a second architecture, even a simpler one, gives the students a breadth of experience that prepares them to quickly move to any number of additional processors throughout their career. We have also noticed that the severe constraints of an 8-bit processor require more programming skill from the students, sharpening their skills.

Fourth, the IBM-PC is a relatively complicated hardware environment. DRAM refresh, DMA support, timers, multiple interrupt sources, video, disk, and keyboard controllers, a large BIOS, DOS -- all of these intrude on a discussion of the most basic interfacing concepts. We have found the 6800 trainer to be far more tractable at this level.

Finally, the system bus is generally not available for experimentation in an IBM-PC compatible computer. Numerous experiments have been designed around the ISA-bus expansion slots. We feel, however, that it is important for students to experiment with the system bus itself.

The Need for a Simulator

The disadvantage of switching processors is that part of the second course must now be consumed with learning the new architecture and assembly language.

The students have the programming skills necessary, and there are no major conceptual hurdles in the new architecture. Nevertheless, practice is necessary to become skilled in the new assembly language. Some programming is required to test the hardware in the various hardware experiments. A weakness in programming is a hindrance to performing the hardware experiments within the time constraints of the class.

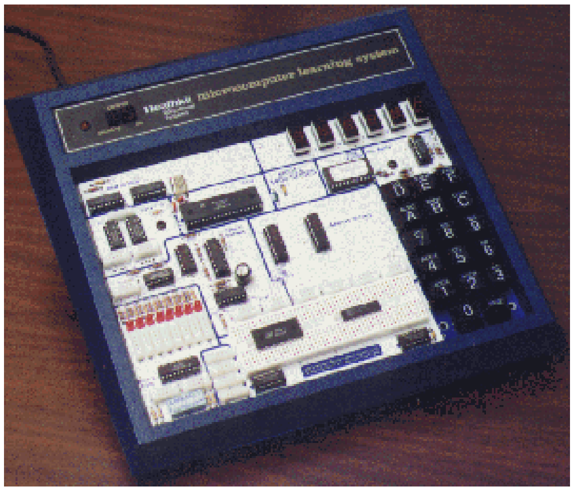
Figure 1. The Heathkit Model 3400A Microprocessor Trainer

Practice, however, is a problem because of the lack of availability of 6800 computers on which to program. 80x86 machines have become ubiquitous, available in campus computer labs and in most dorm rooms. No 6800 machines are available to the students outside the classroom.

6800 software simulators are available. However, they do not duplicate the environment of the Heathkit trainer used in our laboratories. Duplicating this environment is important. It is proficiency *in this environment* that is required to perform hardware experiments in a reasonable time.

For these reasons, we have developed a Microsoft Windows based software simulation of the Heathkit Model 3400A microprocessor trainer (Figures 1 and 2).

Simulation Design

This software was written in *Delphi*, Borland's visual object-oriented Pascal compiler. The visual environment allowed for rapid prototyping and development of the program for a Windows environment. The compiled nature of *Delphi* provides for adequate performance of the simulation, compared to some other interpreted visual languages.

This software is a full simulation of the complete 6800 instruction set. It is more than merely a 6800 simulator, however. The trainer keyboard and display hardware are simulated as well, at the microprocessor instruction level. Students can edit, execute, and test programs exactly as they would on the real trainer.

However, an accurate simulation of the full trainer behavior requires duplication of the Heathkit programs contained in the trainer ROM (much as an IBM-PC compatible computer requires an IBM-PC compatible BIOS ROM).

The Heathkit ROM programs do not contain a copyright notice. Nevertheless, Heathkit has been contacted regarding permission to copy this program. An answer was still pending at the time this paper was written.

The simulated 6800 instructions run at reasonably fast speeds, due to the significantly higher clock speeds of modern PC's compared to the trainer's 1 MHz.

The simulated speed depends on the instructions being executed. A typical test program that runs in 2 seconds on a real 1MHz trainer requires 60 seconds on my 66 MHz 486. This is an effective simulated clock speed of 30 kHz. This has proven to be adequate for student use, and we have not attempted to optimize the code to improve this performance.

We expected that accessing the simulated display, however, would significantly slow operation. When the appropriate store instruction is sensed, the real hardware turns on or off a display segment with no additional software delay.

In the simulator, the display segment must be re-drawn on the screen in its new display state. This requires writes to numerous on-screen pixels, and consumes many processor cycles.

A simple timing loop that counts rapidly on the display demonstrates otherwise. An effective clock rate of 20KHz was measured. This video performance is of course dependent not only on the processor speed but also on the video adapter in use.

Integration in the Curriculum

This simulation is used in Ele 404 Microprocessor Interfacing. The emphasis of this course is on computer hardware. Our chosen platform, for reasons explained previously, is the Motorola 6800. Proficiency in 6800 assembly language is necessary to complete the various hardware assignments and exercises.

The students upon entering this class have completed a semester of 8086 assembly language programming, but have no experience with the 6800.

The 6800 architecture, in comparison to the 8086, is relatively simple with no new abstract concepts involved. The students are expected to master the basic instruction set in the first two weeks of the semester.

Classroom lectures highlight differences between the two architectures. Homework assignments give students practice in writing 6800 assembly language. Two one-hour laboratory exercises introduce them to the trainer, its memory map, and its I/O capabilities.

The Trainer Simulator allows the students an arbitrary amount of practice time with the new language on personal computers on their own time.

Hardware vs. Simulation

All of the hardware laboratory exercises throughout the semester are performed on the real 6800 hardware. We have no desire to implement a full software simulation of the hardware. In our experience, it is important that students be allowed to make the types of mistakes that are possible only with real hardware. As an extreme example, a reverse-polarized capacitor that explodes or bursts into flame is a far more effective teacher than a computer dialog box that informs the student, in cold factual terms, that a component has been inserted backwards.

Nevertheless, in this class lack of familiarity with the microprocessor and the associated operating system (ROM) is an impediment to gaining experience with the hardware. The simulator we have designed allows the students to quickly master the software in an environment very similar to the real hardware.

Enhanced Simulation

A further benefit of the simulation is that we can provide enhanced capabilities to the trainer at minimal cost. For example, the simulator optionally provides a dis-assembler, which the real trainer cannot do at reasonable cost.

As shown in Figure 2, the mnemonic for the opcode being examined is displayed to the right of the keypad.

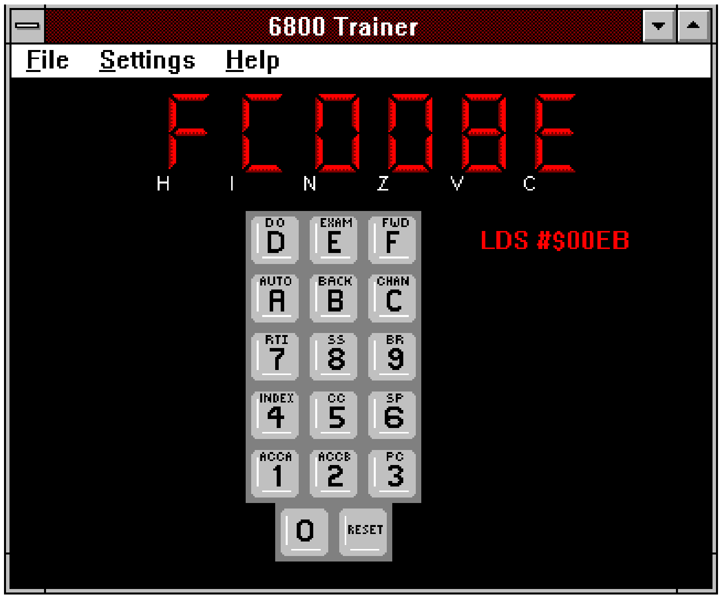

Figure 2. The Simulator

On-Line Documentation

In addition to operating instructions, the simulator provides on-line help containing some of the most commonly used reference material contained in the trainer reference manual and the programmer's reference card. An opcode listing, memory map, and ROM subroutine reference put needed information at the student's fingertips.

Loading of Executable Files

One of the most time consuming aspects of testing programs on the real trainer is the time required to enter the program in hex by hand. Although an assembler may be used to translate the program into opcodes, these opcodes must still be entered one by one. If a program overwrites itself in memory due to a logic error or a typing mistake, the entire program may need to be re-entered repeatedly. A large amount of time is consumed entering, checking, and re-entering programs.

The simulator significantly improves this situation by adding the capability to load programs directly into the simulator memory. Most 6800 assemblers (including the freeware assembler available from Motorola) can produce output in the Motorola standard S19 format. The simulator loads these S19 files directly into memory.

We do require students to perform some hand assembly of programs. In their first exposure to the trainers they do enter the programs by hand, in order to become familiar with the interface. After that, they quickly begin using an assembler.

Note that this feature is not currently available on our trainers. When the students perform the hardware experiments, they must still enter the programs by hand.

Future Work

We have two plans for future improvements in the simulator.

Verifying correctness

The instruction set is not yet fully tested. It is difficult to construct a test suite that exercises every instruction in every addressing mode and checks every possible condition code outcome for every possible operand.

A significant disadvantage of using simulators in general is that the simulation may not be exact. If the simulator contains errors or differences from the real processor, learning is hindered rather than helped. We have experienced this problem with one of the freeware 68000 simulators that is available.

Our simulator does run all of the BIOS functions correctly. The BIOS exercises a large portion of the instruction set, and is a reasonable indication that the bulk of the instruction set is implemented correctly.

We have two plans for further testing: 1) design test programs that systematically test portions of the instruction set. The results of these programs (for example, a checksum of all the calculations) can be compared on the simulator and on a real 6800. 2) Conduct group reviews of the simulation code, verifying the logic and looking for errors.

Simulating interface hardware

The software interfaces to programmable interface chips (such as the 6821 parallel I/O interface or the 6840 programmable timer) can be challenging for students. For reasons discussed previously we have no intention of replacing the hardware experience with these chips. Nevertheless, we are considering implementing a software simulation for the purpose of programming practice.

In our experience the first programmable interface encountered is the most difficult. Once the concept of a programmable interface chip is mastered, the students quickly master details of future chips. We may implement a single interface simulation, such as the 6821, to get them over that initial learning experience as quickly as possible.

Conclusions

This software was used for the first time in the spring semester 1996 at Bob Jones University. No systematic measurements of its effect on student performance have been attempted. Our personal observations and student feedback suggest, however, that it is going to be an effective tool for learning 6800 assembler and the trainer environment.

Links:

- ET3400 microprocessor trainer simulator
- About the simulator