MICROCOMPUTERS IN THE COMPUTER SCIENCE CURRICULUM

Alfred C. Weaver
Department of Applied Math and Computer Science
University of Virginia
Charlottesville, Virginia

IMPACT

The impact of the ubiquitous microprocessor is being felt at all levels of education and industry. It is not only changing the technology of production, but altering the basic concepts of the design cycle itself. The tremendous flexibility of "programmable architecture" [1,2], or "dynamic configuration" [3,4], dictates that hardware-standardized, software-customized modules, of which microprocessors form an integral part, are the way of the future. Changes of this magnitude must be reflected in our computer science curriculum, else our own relevance is suspect. How to reflect this technological shift in the classroom is the subject of this paper.

REALIZATIONS

A first realization is that microprocessor-based systems are, by their very nature, hybrid hardware/software systems. True, microcomputers can teach either hardware or software with only lip-service to the other, but to do so is to abridge the total learning experience available. The electrical engineer who denies the existence or importance of sophisticated programming techniques and cross-software support systems is missing the point just as surely as the computer scientist who disdains the ability to quickly and accurately implement a hardware device interface. The hardware/software nature of microprocessors can be explained, utilized, and even exploited for the benefit of the student who will soon be designing his own microprocessor-based systems.

Secondly, microprocessors are popular. They represent state-of-the-art technology and as such benefit directly from students' natural curiosity and motivation. The first offering of a graduate-level computer science course at the University of Virginia attracted wide-spread attention and inquiry from students, faculty, government, and industry within a 150 mile radius.

A third realization is that microprocessors illustrate some basic computer science concepts as well as, or even better than, a large machine. Techniques of interrupt handling, peripheral device interfacing, direct memory access, assembly language programming, etc., are easier to teach and understand when presented in a simple environment uncluttered by the non-enlightening complexities of a big machine.

Finally, the hands-on experience provided by low-cost microprocessor-based equipment is an invaluable and cost-effective teaching aid. While lectures present a broad range of knowledge, a laboratory experience permits practical application and in-depth examination of the course material at a very practical level.

STRUCTURE

Should microprocessor education be formalized as a new subject, or should the content of existing courses be modified to include this new material? This decision is as much political as it is pragmatic; a given department's investment in equipment or personnel may make the question moot. However, in the absence of extenuating circumstances, a new course offers both the greatest challenge to the designer as well as the maximum potential benefit to the student. Certainly there is no lack of material to fill a one-semester course! See "Course Outline".

Faced with a variety of educational goals and student backgrounds (CS majors and non-majors, engineers and non-engineers, graduate and undergraduate, full-time resident and part-time industrial), can a microprocessor course be structured to provide something of benefit to everyone? A possible solution is the use of variable credit. This technique enables each student to contract for the amount of work he or she is willing to perform in exchange for a proportional amount of academic credit. One option attempted was to structure the course in three parts: required lecture, including homeworks and exams; optional lab; and optional project. Lecture-only was chosen by those who needed or wanted only a survey knowledge of the subject; typical of this category is the student about to graduate in a related discipline who wants to round out his educational background before seeking jobs in industry. The lecture/lab combination was popular with those who found the hands-on experience exciting and challenging, but whose course load that semester prohibited undertaking an additional project. Lecture-project was useful to those who were already working in a related area, or who were contemplating future research work with microprocessors; double benefits were obtained by applying the project portion of the course to a research task. Finally, the lecture/lab/project combination was student-rated as the "most valuable" of the four options because it provided such an intense learning experience. As expected, it also demanded considerable (perhaps too much) time and dedication on the part of the student.

The remaining sections of this paper summarize three semesters of experience in teaching a microprocessor course at the University of Illinois and the University of Virginia. Suggestions and observations are made concerning course content, homeworks, laboratory exercises, projects, and a suggested course outline.

CONTENT

Lecture content can and should be varied to accommodate subjects of mutual instructor-student interest. One set of topics which adequately (perhaps abundantly) filled 45 lecture hours included the following.

(1) Motivation. The first two lectures consisted entirely of examples of contemporary microprocessor utilization. Tying the course immediately to real life set a tone of relevance which enhanced student motivation.

(2) In-depth example. Next was the intensive presentation of one microprocessor (Intel 8080), including hardware characteristics, instruction set, available software, and architectural advantages/disadvantages. The chip was examined from the inside out, starting with its finite-state machine design and working outward, through its instruction set, to its real-world interface signals. A simple microcomputer system using minimal components (microprocessor, clock, bus driver, memory, and I/O port) was designed on the blackboard.

(3) Manufacturer-supplied software. Everyone in the course would eventually program a microprocessor and either simulate via interactive cross-software (in homeworks) or observe (in lab) the execution of their programs. Some time was spent describing the peculiarities of the main cross-software provided by the manufacturer. For the 8080, a discussion of MAC-80, INTERP-80, PL/M, and library programs was appropriate.

(4) Course projects. Two additional lectures were devoted to discussion of acceptable course projects. Allowable projects included hardware design, software implementation, language design, and library research. Each student was allowed to pick one of the proposed projects or suggest his own. No student was allowed to continue the course until his project proposal had been edited sufficiently to gain the instructor's approval. This is of particular importance when specifying how much hardware and/or software, or how much design and/or implementation, is finally required.

(5) Comparative architecture. Other microprocessors were examined as in (2), but in less detail to prevent boredom. If the 8080 was used in (2), then a look at two or more of the Motorola 6800, MOS Technology 650x, Zilog Z-80, or TI 9900 would be instructive.

(6) Support software. Similarly to (3), we examined the manufacturer-supplied resident and cross-software supplied for each machine presented in (5). Also introduced at this point was the concept of "universal" cross-software, including assemblers and simulators [5,6]

(7) Microprogramming. The theory of microprogramming [7] was covered, followed by its implementation in the micro world (e.g., Intel 3000, AMD 2901). A case study of its use in a multiple-precision arithmetic unit was presented.

(8) Real world applications. Applications, as in (1), were re-introduced here to reinforce motivation after mid-semester. Also, details of implementation can now be expanded. At this point the student has enough background to appreciate some technical examples, such as microprocessor uses in industrial process control, electronic navigation, and medical instrumentation, and can appreciate a case study, such as replacement of discrete logic with microprocessor software in a peripheral device controller.

(9) Automatic generation of software. We briefly investigated and discussed the formal techniques for describing digital systems and current progress toward automatic generation of assemblers, simulators, and loaders from an architectural description [5,6].

(10) Put it all together. A case study of a real world design problem is appropriate as a final topic. Important considerations include the specification of the problem itself, the selection of an appropriate microprocessor and support hardware, the actual system design and interface with existing equipment, production of the resident software, and the hardware/software trade-offs encountered during the design phase.

HOMEWORKS

Homeworks should enhance subjects introduced in lecture or encourage independent exploration of areas not suitable for, or time limited by, a lecture-style presentation. Some suggestions:

(1) Research report. Conduct a library search for documentation on a current use of microcomputers. Examples: point-of-sale terminals, home appliances, video games.

(2) Hardware design. Design a microprocessor-based system to recognize sequential input from a numeric keypad and display same on a seven-segment LED display.

(3) Software design. Write the software which would turn (2) into a running machine. Assemble and simulate using supplied cross-software.

(4) Hardware replacement. Given the functional description of a simple hardware module, write a functionally equivalent software package (subroutine) which could replace it.

(5) Comparison of assembly language vs. high-level language. Given a simple software task, implement its solution first in assembly language (e.g., MAC-80), and then in a high-level language (e.g., PL/M). Gather statistics on program development time, program debug time, efficiency of the generated code, memory space required, and execution time of the final code. Draw conclusions concerning the conditions under which each type of programming is appropriate.

PROJECTS

Projects could be chosen largely at the student's discretion. One month into the semester a contract was signed between instructor and student which clearly specified what was to be accomplished. Projects, depending upon complexity, could be accomplished individually or in teams of two (maximum). The acceptable level of complexity for a project was a function of the student's level (graduate, undergraduate) and his background. Thus, the selection of a project was a highly individual process. Some of the more interesting projects completed included:

(1) traffic control system
(2) audio cassette interface
(3) acoustic digitizer
(4) building an 8080 system from the chip set
(5) interfacing a KIM-1 to a Baudot TTY
(6) design and construction of a serial I/O card
(7) design of a high-level language especially for micros
(8) implementation of a microprocessor-controlled music box
(9) floppy disk controller (2 semesters)
(10) software for floppy disk filing system
(11) floating point software package
(12) controller for heating/cooling system
(13) graphic display controller
(14) design of an "optimal" microprocessor instruction set
(15) implementation of a Z-80 assembler and simulator
(16) anesthesia monitor

LABORATORY

The lab was the vehicle for implementing topics first introduced in lecture. The hardware support necessary is an assembled microcomputer system (e.g., IMSAI 8080) or an equivalent locally-developed system (e.g., MUMS [3]). Emphasis was on gaining familiarity with microprocessors, learning their software, and interfacing their hardware (in that order). Students working in teams of two in an open lab (lab stations were scheduled for convenience but no minimum or maximum number of hours was imposed) were able to accomplish the following six exercises:

(1) Write a simple program to create a specific display pattern (e.g., a left-shifting 1) on an output LED register. Assemble the code by hand and load the program in binary using toggle switches.

(2) Repeat (1) using a supplied cross-assembler and a supplied monitor for loading.

(3) Write an absolute loader for object code on paper tape. Include a simple teletype monitor for commands (start, stop, restart) and check-sum error detection for each load block.

(4) Design a teletype software interface using double buffering. Implement it first using polling, then repeat with an interrupt-driven scheme.

(5) Design and build the hardware necessary to interface the microprocessor to a Tektronix display, using two D/A converters. Demonstrate correctness by running the hardware with instructor-supplied software.

(6) Design and implement the software package necessary to accept line definitions (two (x,y) endpoints) from a TTY and display that line on the CRT screen. Add a simple display file manager to permit definition and display of figures containing multiple lines.

COURSE OUTLINE

I. Architecture of microprocessors

1. Characteristics of microprocessors
2. Limitations implied by (1)
3. Memories (RAM, ROM, PROM)
4. Support chips (e.g., clocks, bus drivers, buffers, UARTs, interrupt and DMA controllers)
5. Survey of some currently available microprocessors (e.g., Intel 8080, Motorola 6800, MOS Technology 6502, Texas Instruments 9900, Zilog Z-80, Intel 3000, AMD 2901)
6. Interrupt structure
7. Direct memory access
8. Complete vs. bit-slice CPUs
9. MOS vs. bipolar technology
10. Input/output facilities, capabilities, and limitation

II. Software Systems

1. Software constraints due to time and space
2. Manufacturer-supplied ROM utility programs
3. Cross-assemblers and cross-compilers
4. Designing microcomputer software systems
5. Case study: a "universal" assembler
6. Programming microcoded systems

III. Architecture of microcomputer systems

1. Peripheral interfacing
2. Communications
3. Configuring a bare-bones system
4. Configuring a modular system (MUMS)
5. Configuring a network
6. Microprogramming
7. Matching the architecture to the application
8. Choosing a "home" computer system

IV. Real world applications

1. Replacing random logic with software
2. Industrial process control
3. OEM equipment
4. Mass markets

LABORATORY

Lab exercises will stress: cold start of a bare-bones system; input/output; communications with peripherals; communications protocols; peripheral device hardware/software interface; loaders; downloaders; writing and running assembly language programs; high-level languages for microprocessors; comparison of assembled vs compiled code.

PROJECT

The course project is expected to be a non-trivial software, hardware, or hybrid project, chosen individually by each student and approved by the instructor. Projects may be cross-software systems (e.g., assembler, emulator, definition of an intermediate text for a "universal" assembler), resident software systems (e.g., peripheral device software interface, inter-processor communication, new video game), or hardware development (e.g., hardware peripheral device interface, design of new microprocessor, implementation of an industrial process control problem).

OBSERVATIONS

The course style (lecture, homeworks, exams, lab, and project) was very well received and highly praised by student evaluations. The course content was judged "exceptionally good" overall, but suffered more variance in the ratings due primarily to the diversity of the audience. For instance, third-year undergraduates were "snowed" by the introduction of microprocessor applications to industrial process control, while Ph.D. candidate graduate students were bored by the discussing of microprogramming, a concept which they already understood well. Limiting the diversity of the audience, particularly by strict enforcement of the prerequisites, seems to be the only practical solution to this problem.

Persons taking the "full load" (homeworks, exams, project, and lab) rightfully complained of overwork; all, however, were excited by the intense experience. Students desiring only a survey knowledge reported the accomplishment of that goal with a minimal investment of time outside the classroom. The lab experience was unanimously reported to be an essential ingredient for thorough understanding. Most students who undertook a course project also elected to participate in the lab. Using teams of two in the lab (primarily a concession to the amount of hardware available) reduced the individual workload somewhat without any apparent degradation in the learning experience.

No textbook was used since no one book contained all the desired subject matter; instead, a large number of handouts throughout the semester provided lecture documentation. While the use of handouts is necessary to supplement weakly-documented areas, one or two carefully selected textbooks would have eased the burden of preparation. Several appropriate textbooks [7,8,9,10,11] are now

available.

REFERENCES

(1) Weaver, Alfred C., "A Graphically-Programmed, Microprocessor-Based Industrial Controller," Ph.D. Thesis, Department of Computer Science report no. UIUCDCS-R-77-865, University of Illinois, Urbana, Illinois 61801, May 1977.

(2) Weaver, Alfred C., "A Graphically-Programmed, Microprocessor-Based Industrial Controller," Proceedings of the Rocky Mountain Symposium on Microcomputers, Fort Collins, Colorado, August 1977, p. 240-260.

(3) Faiman, Michael, Weaver, A. C., and Catlin, R. W., "MUMS - A Reconfigurable Microprocessor Architecture", IEEE COMPUTER, January 1977, p. 11-17.

(4) Faiman, M., Catlin, R. W., and Weaver, A. C., "A Modular, Unified Microprocessor System (MUMS)," Proceedings of the DISE Workshop on Microprocessors and Education, Fort Collins, Colorado, August 1976, p. 1-5.

(5) Kominczak, Charles P., "A Universal Cross-Assembler", Department of Computer Science report no. UIUCDCS-R-76-803, University of Illinois, Urbana, Illinois 61801, May 1976.

(6) Johnson, Gearold R., and Mueller, Robert A., "Automated Generation of Cross-System Software for Microcomputers", IEEE COMPUTER, January 1977, p. 23-31.

(7) Tanenbaum, Andrew S., Structured Computer Organization, Prentice-Hall, 1976.

(8) Hilburn, John L., and Julich, Paul N., Microcomputers/Microprocessors, Prentice-Hall, 1976.

(9) Soucek, Branko, Microprocessors & Microcomputers, John Wiley and Sons, 1976.

(10) Peatman, John B., Microcomputer-based Design, McGraw-Hill, 1977.

(11) Klingman, Edwin E., Microprocessor Systems Design, Prentice-Hall, 1977.