Teaching Computer Systems to Majors: a MIPS Based Solution

Murray Pearson, Tony M[c]Gregor and Geoffrey Holmes

Abstract

The debate over whether a Computer Science graduate should have an appreciation of hardware related topics has a long history in Computer Science Education. Given that the subject is getting broader and specialization is occurring earlier in a Computer Science degree the calls to omit this vital aspect of the subject are stronger than ever. One reason for avoiding this type of material is that its *flavor* is not to every student's taste, especially students interested in Information Systems as their major. If it is considered an essential component of a major then the challenge to educators is to produce a course that is technical without technical detail dominating essential concepts, coherent from a systems' perspective and relevant to majors who might have quite different orientations within the subject.

I. INTRODUCTION

The ACM guidelines for an undergraduate program in computer science [1] and Information Systems [2] both include coverage of computer architecture, logic design, operating systems and computer communications as core material that all students should meet. For many students this portion of the curriculum may well be the final chapter in their hardware education, especially if they are in a programme that involves a degree of specialisation. This situation generates a significant challenge for the teacher of such a course. Not only do they need to stimulate those students who wish to continue to study hardware related subjects, but they must maintain the interest of students who do not have an *a priori* interest in hardware. For these students it is important to emphasise the logical equivalence of software and hardware, and to show them a coherent view of a computer system so that they can more fully appreciate the relationships between the different components of such a system. This has to be achieved without falling into the trap of describing too much device-specific detail. Technical detail in a computer systems course cannot be avoided, of course, but its effects can be minimised by a careful selection of a supporting architectural model.

This paper describes the development of a new computer systems course and a platform to support teaching of the course in the Department of Computer Science at the University of Waikato. The course is a compulsory course for all computer science majors and includes the full range of students from those taking an information systems stream to those taking a computing technology stream who will take further courses in Computer Architecture.

Despite the diverse needs of these students the course has achieved a substantial success. This has been achieved through the use of real hardware and by integrating computer architecture with operating systems and data communications

The authors are adjuncts to The Department of Computer Science University of Waikato, Hamilton New Zealand. Email: mpearson@cs.waikato.ac.nz

II. COURSE OUTLINE

The departments curriculum committee established a set of key topics that should be covered by the course. These included data representation, machine architecture (including assembly language programming), memory and I/O, operating systems and data communications.

The initial planning of the structure of the course was conducted through text selection. Popular texts such as Tanenbaum's Structured Computer Organization [3], Stalling's Computer Organization and Architecture [4], and Patterson and Henessey's Computer Organization and Design [5] were considered to be too technical for a core computer systems course. While no text has been found that covers all of the material in the course the text "A Programmers View of Architecture" [6] covers the material in the first half of the course very well and has been used for the last two years.

III. SELECTION OF THE MODEL ARCHITECTURE

The first stage in selecting a new platform was to decide whether to use a simulated or a real architecture. Using a simulated system offers two main advantages. Firstly, it is possible to develop a simulator to any desired CPU. This is an advantage as it is then possible to develop a CPU that is tailored to the goals of the course. Secondly, using a simulator offers a number of possibilities to generate visualizations of a program executing which can be used to help reinforce important concepts. While using a simulator offers advantages, it is itself a program running on a computer. This makes if difficult for students to readily identify the target system and tend to confuse the roll of components of the system. When this happens there is a risk that students will focus on the most obvious difference between practical work in this area and others: the programming language. When real hardware is used the real focus is more likely to be on the target system. We strongly believe that this disadvantage far outweighs the advantages of using simulation. For this reason we have chosen to go against the trend and use a real system.

Because the goal of the course is to explain the role and interaction of the components of a computer system not to teach assembly language programming for its own sake there are two main requirements for a model architecture:

1. a simple, easy to learn instruction set
2. an architecture that can easily demonstrate the relationship between high and low level languages.

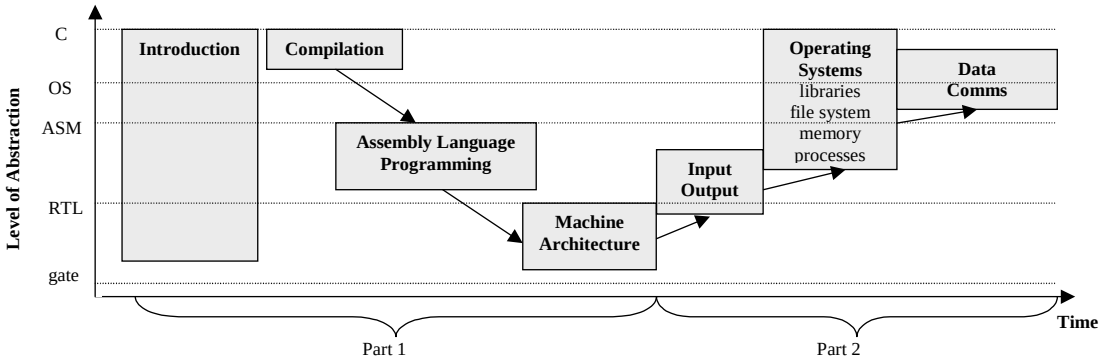

Figure 1 Topics Covered in the Course

For these reasons simplified MIPS architectures are used in both the Goodman and Millar [6] and the Patterson and Hennesey [5] books. In both of these cases however, they have elected to use simulators to support the practical components of their courses. However for the reasons described above we have chosen to use a real MIPS based system.

While the MIPS processor is suitable for teaching assembly language programming there are a small number of features which were included to improve its efficiency. The most noticeable of these are semantics associated with load and branch slots that only add complexity when used in an educational environment. To cope with both of these situations the students are instructed to place “nop” (no operation) instructions after each load or branch instruction. We have found this to be a perfectly satisfactory solution to date. These problems were considered to be small enough that their disadvantages did not outweigh the benefits of using a real system comprising a MIPS processor. In the future we plan to develop an assembly pre-processor that will notify the students if a nop is not placed after either a load or branch instruction.

IV. COURSE CONTENT

Figure 1 shows the order of the topics that make up the course and the relative levels of abstraction used to describe them. The main content of the course can be broken into two parts. The first part illustrates what happens to a high level program when it is compiled and executed on a computer system. This serves two purposes. First it demonstrates some of the major issues which determine the performance of a computer system. Second, it shows the likely consequences of writing a particular construct in a high level programming language in terms of speed and size of the code generated.

To gain a good insight into this process the student must have a good understanding of assembly language programming and machine architecture. These form the next two major topics covered in the course. As programming at the assembly level is a foreign concept to most students entering the course, an attempt is made to introduce it using examples from the C programming language (a language they are familiar with). Not only does this add interest and illustrate the motivation for particular programming constructs, it also shows the relationships between a high level language and the corresponding assembly language. To complete this section the students do an assignment where they document the assembly language code generated by the gcc compiler.

Once the students have a good grounding in assembly language programming they complete a section on the assembly process and machine architecture again using MIPS examples. To illustrate how machine language instructions execute on hardware, a simple MIPS architecture is introduced as shown in Figure 2. Each component of this architecture is introduced with the control signals used to control them. To reinforce the process, we use interactive sessions where the class pretends to be the control unit for the CPU and a number of example instructions are fetched and executed on it. While this architecture is significantly simpler than an actual MIPS CPU it illustrates the important issues that determine the performance of a particular architecture. It also forms the basis for students to move onto more advanced computer architecture courses.

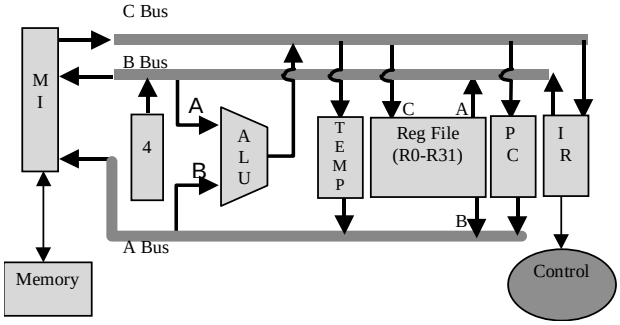

Figure 2 Example MIPS Architecture

The aim of the second part of the course is to produce an understanding of operating system principles and components, their role in supporting the user, and in the execution of programs written in high level languages such as C (the starting point of the course). The focus is on achieving an empathy with the operating system rather than an ability to write a new one.

This empathy is achieved by building on concepts introduced in the first half of the course starting with the

introduction of I/O. The concept of interrupts are then introduced as a means of making more efficient use of systems resources. This leads into multitasking, memory management, and the typical system calls provided in modern operating systems.
Most modern computer systems communicate with other systems. The final section of the course is aimed at giving an understanding of data communication principles (low-level) and networks (high-level).

V. PRACTICAL COMPONENT

To support the practical component of the course a MIPS based system has been designed and manufactured. In keeping with the flavor of the course we wanted to use a minimal system that only used components used by the course, and we also wanted all the components of the system to be visible to the students. For these reasons we decided that the system should be contained on a single printed circuit board (PCB).

A search of suppliers did not reveal any system that matched the requirements for the course. So a board was designed to support the course. In designing the board a number of requirements were set to make it more suitable for the teaching environment. The first was to ensure that the board was laid out so that it was easy to identify all of the major components that make up a computer system (i.e. processor, memory, bus and input/output). The second was to include a set of switches and a seven-segment display in the design which programs running on the board could access, giving students a greater sense of using the actual hardware. To protect the boards and let the students see all of the hardware, the boards have been packaged in a case with a clear perspex cover. A sticker has been attached to this cover to identify the major components.

The board contains two serial ports. One is connected to a PC running LINUX and provides an interface for the students to interact with the board. The second is connected to a dumb terminal and is used for the I/O assignments.

The assignments that make up the practical component of the course are shown in Table 1. Of particular note is the implementation of a multitasking kernel by the students. Given that most students are not computer technology students and that most successfully complete this exercise we belief this is a major indication of the success of the course.

Each assignment has been arranged so that there are a number of finishing points allowing more advanced students to be extended without overwhelming the less able students.

VI. CONCLUSIONS

In light of our experiences, we believe that this approach to teaching computer systems has great merit.

Firstly, technical detail is kept to a minimum by the model architecture that we have chosen. It would seem from the student feedback and their performance in the course that this model is appropriate when dealing with a broad spectrum of student interests.

Secondly, the MIPS system with its single well laid out PCB (with no superfluous components) that has been developed provides a clear, reliable, and realistic vehicle for teaching computer systems.

Finally, the course does not seem to discourage non-computing technology students who see real value in taking the course. These students take the course seriously and this in part is due to the open-ended nature of the assignments which allow motivated students to be extended.

Assignment Topic	Objectives
Data Representation	Write program to read in different types and out memory location contents
Introduction to MIPS board	Assembling, downloading, executing and debugging programs written for the board
MIPS programming I	Write a series of assembly language programs to read values off the switches and display on seven segment display
C -> MIPS	Write a C programming and compile it to assembler and document the assembler showing the relationship between the two programs
RTL design	Paper exercise to simulate the execution of a number of MIPS instructions on a model architecture
MIPS programming II	Write a program to set up and use a serial I/O device
MIPS programming III	Write a program to set up an interrupt service routine
MIPS programming IV	Write a multitasking kernel to allow switching between a fixed number of tasks
MIPS programming V	Write an extended multitasking kernel
Data Communications	Investigation of simple error detection mechanisms

Table 1 Assignments for Computer Systems Course

VII. REFERENCES

[1] ACM Curricula Recommendations Volume I: Computing Curricula 1991: Report of the ACM/IEEE-CS Joint Curriculum Task Force. ACM 1991.

[2] IS'97 - Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems. ACM 1997.

[3] Tanenbaum, A.S., Structured Computer Organisation (Third edition), Prentice Hall, 1990.

[4] Stallings, W. Computer Organisation and Architecture: Principles of and Structure and Function (third edition), MacMillian, 1993.

[5] Patterson, D. A. and Hennessy, J.L. Computer Organisation and Design: The Hardware/Software interface, Morgan Kaufman, 1994.

[6] Goodman, J. and Millar K., A Programmer's View of Computer Architecture with Assembly Language examples from the MIPS RISC Architecture, 1992.