A lab course of Computer Organization

J. Real, J. Sahuquillo, A. Pont, L. Lemus and A. Robles
{jorge, jsahuqui, apont, lemus, arobles}@disca.upv.es
Computer Science School
Department of Computer Engineering
Technical University of Valencia (Spain)

Abstract

Lecture topics in Computer Organization courses offered by different Universities around the world do not differ significantly. This is because, in general, lecturers use the same textbooks and are inspired by common curriculum sources. However, lab courses and project assignments require more and more expensive resources (computers, assemblers or assembler simulators, logic circuit simulators, …) This fact, together with the rapid advance of these tools, causes lab courses to widely differ among universities.

This paper summarizes the lab course on Computer Organization offered this year at the Technical University of Valencia, Spain. The course is composed by several experiences and jobs, each one aimed at working on one specific topic. Our goal is not only to introduce the tackled topics, but also to discuss some characteristics of the tools. All the tools used are freely available, which is a must for the students to be more motivated and to be able to extend their work using their own computers at home.

1. Introduction and motivation

The Technical University of Valencia offers a three-year Bachelor degree course in Computer Engineering. A modification of the curriculum has recently been undertaken to adapt it to the new trends and professional outlines. The recommendations from the IEEE/ACM Computing Curriculum 2001, as well as curricula from some relevant Spanish and foreign universities have influenced the new design. The course includes 60 lab hours (25% of the total), distributed along two core courses in the first and second year. Each course is attended by more than 800 students, which strongly impacts on the lab organization and the type of experiments. Up to 40 students attend each lab session, working in groups of two people. Theoretical lectures are attended by up to 120 students.

To properly design the lab course it is necessary to consider the contents of the theoretical courses, the academic year when they are given and, specially in our context, the high number of students, which this is not a trivial task. One of the main problems is to choose appropriate tools for the lab experiences. An excessive use of abstract simulators is a risk because some of them (specially those very *didactic*) are quite far from the real world. On the other hand, the contents of the Computer Organization subjects are very difficult to implement in a practical way without additional technical knowledge. Finally, the tools and equipment needed for the lab sessions tend to be expensive.

Some universities propose lab courses based only on a part of the subject (generally the part whose contents are easier to practice in the lab) and they do not cover, in a practical way, the whole theoretical contents. The main reason is usually the lack of appropriate tools to do it.

The structure of this paper is the following: section 3 briefly describes the theoretical course of Computer Organization; section 4 details the lab course, both describing the experiences and the needed tools; section 5 presents the time schedule of the theoretical and lab courses. Finally, section 6 summarizes our conclusions.

2. Computer Organization theoretical course

The Computer Organization course is a core subject of the Computer Engineering degree. This course is given along the first and second year of the degree, having assigned up to 180 lecture hours in all (90 lecture hours each year). Evaluation is performed in an annual basis.

The main goal of this course is to introduce the students to the organization of computer systems through the study of each one of the functional units that compose them. Topics include data representation, digital logic, assembly language, simple processors, memory unit, input/output unit, arithmetic-logic unit, basic pipelining, and advanced processors.

Tables 1 and 2 show the themes into which each topic is broken down and the number of hours assigned to them. This information corresponds to the syllabus of the first and second year courses, respectively.

Syllabus (First year)

Topic	Themes	Hours
Introduction	1. Introduction to computer systems	2
Data representation	2. Data representation	9
Digital logic	3. Basic concepts of digital systems	12
	4. Combinational systems	10
	5. Sequential systems: Flip-flops	4
	6. Sequential systems: Registers and counters	8
Assembly language	7. Introduction to assembly language	10
	8. Assembly programming	6
	9. Procedures	6
Simple processors	10. Datapath	10
	11. Control unit: Hardwired realization	8
	12. Control unit: Microprogrammed realization	5
	Total hours	**90**

Table 1. Syllabus of the first year course on Computer Organization.

Syllabus (Second year)

Topic	Themes	Hours
Memory unit	13. Memory system	3
	14. Memory system design	10
	15. Memory hierarchy	10
Input/Output unit	16. Input/output devices	9
	17. Input/Output management	12
	18. Buses	4
Arithmetic-Logic unit	19. Integer arithmetic unit: Adders and subtracters	6
	20. Integer arithmetic unit: Multiplication and division	8
	21. Floating-point arithmetic unit	4
Basic pipelining	22. Introduction to the pipelining	6
	23. Pipelined processor	12
Advanced processors	24. Examples of contemporary processors	4
	25. Introduction to multiprocessor systems	2
	Total hours	**90**

Table 2. Syllabus of the second year course on Computer Organization.

3. The lab course

We propose a selection of experiences on Computer Organization, aimed at covering the classical computer functional units: processor, memory, and input/output system. The lab course goals complement those of the classroom course. We have designed and selected some experiences, trying to balance the course time among the mentioned functional units according to their importance. The aim is to acquire an elementary but complete knowledge about Computer Organization as well as its basic working principles and underlying design aspects. We also discuss the selection of a set of free software tools that allow those students requiring additional time, or those who show further interest, to continue their work at home.

The described experiences are organized in lab sessions, each taking two hours of work.

3.1 Experiences

Experience 1: Assembler

Three lab sessions are dedicated to implement simple assembly language programs. The topics are assembly instructions (bare machine) and pseudoinstructions, instruction coding, data representation, and functions in assembly language, exercising the MIPS register usage convention.

The first session is an introduction to the PCSpim interpreter [spim02] that simulates how the assembler works for the MIPS architecture. The session lab is addressed to give the students practice with several features of the tool, and to strengthen some topics studied at the classroom, like character, integer and floating-point representation, as well as memory data alignment.

The second session has three types of exercises. The first one deals with the instruction coding. Students must codify some assembly language instructions and check if their results match to those given by the tool. The second one is addressed to check the results of some instructions that use predefined target registers (e.g., LO and HI for integer division and multiplication instructions). The last one is addressed to running a program that performs the scalar product of two vectors. Students must run the program and answer some questions: i) to determine which function it performs, ii) to identify the pseudoinstructions of the program, and iii) to explain why the assembler not always codifies a given pseudointruction by using the same machine instructions (e.g., the load address instruction).

In the last session, the students must break down the scalar product program in two parts: main program and procedure. The programs must be implemented by using the *callee-saved* as the *procedure call convention.*

Experience 2: The Processor

Three lab sessions are dedicated to the study of the central processing unit (CPU). The main goal of these sessions is to develop a simple CPU (no pipelining) that executes a reduced instruction set - a subset of the MIPS architecture [Patterson97]. The different CPU elements are interconnected by means of busses. The instructions include several arithmetic and logic operations, load and store, and different types of branch instructions, including unconditional, conditional and jumps to subprograms. These instructions permit to implement simple, though fully operating sample programs that can be traced during their execution, allowing the student to follow their steps in the datapath and the activation of the relevant control signals. We use the Xilinx schematic editor and functional simulation tools to implement and test the resulting circuitry [Xilinx01].

The first session is an introduction to the tool itself, as this is the first time it is used. During this session, a register file is implemented and tested. It takes a long time to develop the whole register file, therefore an almost complete version is supplied for the students to complete and test it, according to a set of predefined experiments. The second session deals with a complete datapath, including a Program Counter, Arithmetic and Logic Unit, the memory interface and several auxiliary registers and very simple operators like fixed shifters and a sign extender. Most of these units are supplied in advance and the work to do consists in interconnecting units and testing the resulting datapath by executing isolated instructions. The third session completes the CPU implementation with a Control Unit (CU). It is based on a phase counter and the needed combinational logic to generate the 24 control signals required by the datapath. The students are required to complete the design of the CU by implementing a couple of control signals and then put it together with the datapath. The memory circuit contains a simple program with a loop that has to be tested.

Experience 3: Memory Design

This experience is organized in three sessions. The common goal of all is them is to understand how the memory system in a computer is designed, from the basic cell to the construction of memory modules based on smaller elements and including the decoding and selection system. For this purpose we use the simulation environment Xillinx as tool.

This first lab session deals with the internal structure of memory circuits. The students must design a small memory unit (16x1 bit). We propose this small size for practical reasons: the memory structure designed is also valid for larger memories; the only difference is the number of elementary cells and the size of the decoding circuits.

In the second session, we give the students a predesigned 32Kbytes RAM element, in order to build a 256 Kbytes memory module. The students must pay special attention to access different types of data (bytes or 16 bits words). For checking purposes we supply a module that acts like a CPU, generating addresses and byte selection lines.

In this session, we supply a circuit that simulates a memory system composed by 4 different modules and a checking element that acts as an address generator. With all these circuits the students must implement different memory maps.

Experience 4: Cache Design

The goal of this session lab is to understand why cache memories are the basic and ineludible mechanism that computers incorporate to reduce memory accesses latency.

We give the students a small testing program written in C language (in similar manner to D. Patterson [Patterson01]), to experimentally determine the parameters of the computer's caches.

To perform the experiments the program defines an array of 1 mega integer elements size, and different scenarios are modeled. Each scenario is determined both by the amount of elements that are accessed (1K elements, 2K elements, …) and by the stride (1, 2, 4, …, 512K). The program has a main loop that runs repeatedly many times in which the elements of the scenario are accessed to measure the data access time. Then, all the resulting times are averaged. The loop execution time is relatively long (approximately 1 second) in order to get precision in the measure process.

From the results, the students must firstly notice the number of cache levels Then, for each cache level they must determine: i) the block size, ii) the set associativity, iii) the cache size, iv) approximately how fast the cache hit is, and v) approximately how fast the cache miss is. Some other parameters about the memory hierarchy like the page size and the page fault penalty are also determined.

Experience 5: The input/output system

The main objective of this experience is to practice the basic methods of synchronization: status checking (polling) and interrupts. To achieve this, the students develop simple interactive programs by using the input/output available facilities.

In the first session, we present a hypothetic case of communication between a MIPS R2000 processor and two basic I/O devices: the keyboard and the printer. A simulator acts like these two devices mapped in memory positions. Both are character-oriented devices. The PC keyboard is used as the input device while data output is displayed in a window that simulates the printer. The students must write a small program in MIPS R2000 assembly language to read characters from the keyboard and print them in the printer. The program must use polling for synchronization and program-controlled for data transfer.

In the second part the students have the opportunity to practice interrupt handling in a real computer (PC compatible). They also can access the PC memory and I/O maps. We propose them two typical problems to solve: first, students must modify some of the system interrupts (clock and keyboard are the proposed ones) writing the appropriate routines to handle them. In a second step, they must extend the service given by an existing interrupt handler by linking the system routine with their own handler.

Experience 6: Circuits to Support Integer Arithmetic

The main objective of this experience is to design simple integer arithmetic circuits and to modify them to achieve better performance by using pipelining techniques. This experience is organized in three sessions. In the first one, the students must implement a 16 bit adder/subtracter for integer numbers by using 4 bit carry lookahead adders (CLAs). The basic circuits (half and full adders) that form the CLA must also be implemented. Next, they develop a fast multiplier for two 6 bit unsigned numbers by using a Wallace tree. For this purpose, they build and interconnect carry save adders. The last stage of the Wallace tree is built by using the already implemented CLAs. To complete the fast multiplier, the students must build a partial product generation circuit that takes the two integer operands as inputs and generates the six partial products to feed the Wallace tree. Finally, they have to split this multiplier circuit into pipeline stages. For this aim, the students must identify the pipeline stages and establish the suitable clock period to improve the circuit speedup. The students must simulate and measure the response time. Moreover, they must calculate the speedup the pipeline achieves.

Experience 7: Pipelined Processor

The goals pursued in this lab session are to understand the concept of pipelining, identify hazards, realize how hazards affect performance, and to know how the different solutions for conflict solving are implemented.

A program that simulates the behavior of a pipelined DLX processor [DLXide] is used. The DLX processor [DLX02] exhibits a similar architecture to that of MIPS. In the simulator, instruction execution can be tracked in a time diagram, cycle by cycle, therefore it allows to follow their walk through the different stages. The simulator permits also to define a particular technique for hazard solving, including bubble insertions, forwarding, predict-not-taken branches and delayed branches. The datapath (shown by the simulator) appears modified according to the technique applied. Control signals, memory and register contents and some statistics are also made available by the simulator, which permits to extract some conclusions based on quantitative data.

A simple but illustrative assembler program is supplied for the students to trace its execution in the pipelined datapath. First, they must solve dependencies by inserting bubbles and then counting the resulting CPI. Secondly, more effective techniques such as forwarding and branch prediction are exercised, allowing to observe how these techniques work and to compare results with the previous experiments.

4.2 The tools

For the experiences described in the previous subsection, we are currently using different tools. Below, we briefly describe how we use them and how they allow us to reach the goals of the lab experiences.

1. **Logical board**. It is basically a circuit board with some logic gates and flip-flops that can be interconnected by means of wires and connectors. The board also allows for commercial integrated circuits to be added, thus increasing the number of different exercises that can be tackled. By using real circuits and wires, the student realizes the difficulties in implementing real circuits (bad connections, collision of outputs, etc.) which are more difficult to detected when logical simulators are used. The logical board is used for the most basic circuits, leaving the complex ones to be simulated.
2. **MIPS simulator PCSpim**. For assembly language experiences, we use this free MIPS simulator to implement and trace simple programs. The simulator is complete enough for the intended purposes and makes it straightforward to work in assembly language without having to deal with particularities of the platform. On the other hand, it represents an important economical saving, as PC's are available in all of our labs, differently to MIPS-based computers.
3. **Xilinx schematic editor and simulation tools**. The Xilinx Foundation is an application framework for programming logical devices with logical functions of different levels of complexity, from very simple combinational functions to virtually any larger project with both combinational and sequential components, allowing for tristate devices as well as conventional ones. The tool is complex, if used as a whole, but for the purposes of the course, we only need to be able to specify a circuit and to simulate it. The Xilinx tool offers several ways of specifying a circuit, namely a Hardware Description Language, a Finite State Machine and a Schematic Editor. The last one is the most appropriate for our students, since this is the common way of describing circuits in the classroom as well. On the other hand, the simulator is a powerful tool that allows us to track the behavior of the specified circuit in connection with the schematic editor. Despite the complexity of the whole application, our students quickly learn where to click to carry on their work, since the working platform is well bounded from the very beginning of the corresponding lab exercises. This tool has

proven to be very suitable for implementing our simplified RISC datapath and for the control unit as well. It is also used in the exercises related with memory modules.

4. **DLXide** is a simulation tool of the DLX computer. This simulation tool has been developed by lecturers from the Computer Engineering department of the Technical University of Valencia with the aim of providing a suitable environment for performing pipelining experiences. The simulator is able to simulate the pipelining execution unit of the DLX computer in a cycle-by-cycle basis, also showing how the instructions progress through the pipelining stages. For simplicity, it only supports the integer instructions of the DLX architecture. The tool permits to edit, assemble, and execute a DLX assembly program. There exist separate cache memories for instructions and data. User can initialize and modify both machine registers and data memory contents, which are displayed in two separate windows. Moreover, it is possible to display the instruction memory contents and the instruction addressed by the program counter. Through the configuration window, the user can establish the mechanism used for hazard solving among the following techniques: stalls, predict-not taken, delay-slot 1, and delay-slot 3 for solving control hazards, and stalls and forwarding for solving data dependencies. Step-by-step simulation shows how each mechanism solves the hazards. The simulator runs on MS Windows and Linux operating systems.

5. 2. Coordinating theoretical and lab courses

The first and the second year theoretical courses are 30 weeks long organized in two weekly sessions 1.5 hours long, where both theory and problems aspects are lectured. Sessions take place in classrooms of 160 students capacity.

The lab courses have the same duration as the theoretical and their timing must be synchronized. These courses are organized in two types of weeks (A and B), so that the type of the week alternatively changes from A to B and vice-versa. Students must attend to the lab sessions in those weeks they are registered in. Sessions are two hours long every two weeks and take place in labs of 40 students capacity. This has proven to be more suitable than having weekly sessions of 1 hour.

Table 3 shows the planning of the theoretical and the lab course of the first year. Numbers on the top row refers to the week number. Central row shows the planning (theoretical and problem sessions) of the themes. The first 15 weeks focus on the study of both the data representation and the digital logic topics (T3, T4, T5 and T6) mentioned above. Next, 7 weeks and a half are dedicated to the study of both machine and assembly languages. The remaining weeks are addressed to implement a simple datapath and its control unit (both hardwired and microprogrammed). The bottom row refers to the lab sessions. As it can be seen, lab sessions begin at the same time as classroom sessions. Some times; e.g., when studying simple data paths, the lab session starts a little bit before the theoretical topic is studied at classroom. This does not cause any inconvenient, because that time is devoted to study how the tool (Xilinx in this case) works.

Table 4 shows the temporal planning for the second year course detailed above. In this case, no overlap appears between the theoretical and the lab course.

WEEK NUMBER

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	T2		T3				T4				T5		T6	
P 1			P 2										P 3	

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
T7				T8		T9		T10			T11		T12	
P 4							P 5							

Table 3. Planning of classroom and lab sessions of the first year course. Legend: P refers to practical experience and T to topic.

WEEK NUMBER

1	2	3	4	5	6	7	8	9 10	11	12	13	14	15	16
1	T2			T3				T4			T5		T6	
P1								P2		P3				

17	18	19	20	21	22	23	24	25	26	27	28	29	30
T7		T8			T9	T10		T11				'12	3
P3	P4									P5			

Table 4. Planning of classroom and lab sessions of the second year course. Legend: P refers to practical experience and T to topic.

6. Conclusions

In this paper we have presented a lab course on computer organization, and we conclude that a complete course needs the following requirements:

1. A set of tools of a very different nature (assembler, logical circuit simulator, pipeline simulator) to cover the whole theoretical course.
2. It is important that the tools be as close as possible to a professional tool (e.g. we are currently using the educational version of a professional tool.)
3. It is necessary to devote an important amount of time to learn how the tools work, therefore it is important to chose tools also used in other subjects (e. g. the Xilinx framework is used in Logical Design courses too.)

7. References

[Patterson97] D. A. Patterson and J. L. Hennessy, Computer organization and design: the hardware/software Interface, Morgan Kaufmann publishers, 2nd edition, 1997.

[Patterson01] D.A. Patterson, Course CS61C1C: Machine Structures, UC Berkeley, http://inst.eecs.berkeley.edu/~cs61c/fa01/calendar/week13/lab10/, Fall 2001

[Spim02] J. Larus, SPIM: a MIPS R2000/R3000 simulator, http://www.cs.wisc.edu/larus/spim.html, 2002.

[Xilinx01] Jan Van der Spiegel. Xilinx Web page. http://www.prenhall.com/xilinx/, 2001.

[DLX02] Computer Systems Laboratory, FTP Site for Interesting Software, http://max.stanford.edu/max/pub/hennessy-patterson.software/max-pub-hennessypatterson.software.html

[DLXide] P. López. DLXide web page. http://www.gap.upv.es/people/plopez/english.html