THE COMPUTER ARCHITECTURE COURSE IN A COMMON CORE IN COMPUTER SCIENCE AND ENGINEER DEGREES

Alberto José Proença*

The computer architecture subject is part of any course structure on computer related degrees at most Universities. A comprehensive curriculum spawns from computer structure and organisation to concurrent architectures, including performance evaluation.
The approach to this subject - top-down, from programming techniques to hardware issues, or bottom-up, from digital systems to instruction sets - and its contents and duration, vary from course to course, and often from University to University. The order of the presentations may also vary - from the generic concepts to the practice with specific devices, *versus* from programming a microprocessor to the generic concepts. The advantages / disadvantages of these approaches are discussed in the talk.
An attempt to define a common core of topics, from concepts to laboratory practice, is presented here, based on 2 different sources: the results from a Working Party on Advanced Computer Architecture courses (whose goals are to define a common set of topics for the 6 cooperating Universities under TEMPUS), and the author experience in lecturing this subject to students in 2 different courses - Computer Science and Computer Systems Engineering.

1. Introduction

This communication presents the author's experience in the course organisation and lecturing of the Computer Architecture (**CA**) course to two different University degrees: Lic. Informatics and Systems Engineering and Lic. Maths and Computer Science (both are five years courses). The author lectures at a medium size Portuguese University in one of the southern EEC countries, Portugal.

An overview of the main components of a computer related degree opens the communication, pointing out the main differences between a science and an engineering approach to the **CA** subject. Its role and a short description of the main modules are then summarised. The presentation of some relevant issues closes the communication with some open questions.

2. CA in a Computing Curricula

Computing Curricula at European Universities are spread through courses organised in several faculties: science, engineering, education and economics. The first two often

* Dep. Informática, Universidade do Minho, Largo do Paco, 4719 Braga, Portugal

provides the more demanding graduates: on computer science and on computer engineering.

In a typical computer science/engineering degree, the computer related courses usually contain the following main subjects:

- *fundamentals of computing*, which may include Algorithm and Data Structures, Programming Languages, Compilers, Formal Methods of Specification, ...
- *computer engineering*, which may include Digital Systems, Computer Organisation and Architectures, Interface and Communication, Operating Systems, Concurrent Architectures,...
- *systems engineering*, which may include Software Engineering, Data/Knowledge Bases, Information Systems, Man-Machine Interface, Application Specific Topics, ...

A similar framework is also adopted by the USA Academic Institutions, and presented at the Report of the ACM and IEEE-CS on Computer Curricula (Tucker et al, 1990).

The Computer Organisation and Architecture topic establishes the link between a programming language - used in any application development environment - and the physical machine built out of electronic circuits and managed by an operating system. As such, this topic must be in any computer related degree to allow any student to understand how programs are executed in a computer.

3. CA for Computer Science/Engineering Students

A **CA** course looks at the computer as a machine, and its contents should aim 3 main goals:

- to describe the internal structure or organisation of a computer, as a system, and each one of its parts, the CPU, the memory, the I/O interfaces and the communications (*description*),
- to become aware of its internal details, which are required to efficiently run HLL (High Level Languages) and to support a sophisticated operating system (*analysis*);
- to be able to build new computers, either as integrated systems, or designing new chips/boards (*synthesis*).

The first and second parts of the course - the description and the analysis - should always be included in any **CA** course; however, the synthesis is usually more detailed on Computer Engineering courses. The depth of the synthesis part is what makes the main difference between a **CA** course on a Computer Engineering course and other Computer related degrees.

The organisation of a common core on **CA** for several Computer related degrees must take into account the different backgrounds and motivations of the participants. The traditional approach on **CA** - bottom-up, from digital electronics - may prove unsuccessful with sciences or economics students. The common background all these students have since the early stages of there College activities is in Computer Programming: by the end of the first semester at the University they become familiar

with HLL programming. This suggests a better approach to **CA**, once the basic concepts on Computer Organisation are presented: top-down, from a programming language.

A HLL program consists of an *algorithm* - data operations and flow control instructions grouped into modules - and *data structures* - with declarations of scalar and structured data in global/local environments. The compiler interface seems to be a suitable approach to introduce the internal details which are relevant for machine language programming: data representation, instruction formats, addressing modes, data operations, flow control architectural support for procedures/functions. The analysis part of **CA** can be further extended to include:

- support to operating system functions (including interrupt mechanisms);
- acceleration mechanisms, such as pipelining, cache and support for multiprocessing.

The synthesis part of the course is more relevant to Computer Engineering students and it may require some background on digital systems - finite state machines and programmable logic devices. To design new systems the student must know how current implementations work: a course module on microarchitecture and microprogramming, and the evolution into the RISC architectures may easily motivate engineering students, but for the science/economics students the subject should be presented in a broad way, without too much detail on hardware specific implementations.

Two excellent textbooks take this approach to **CA** (Van de Goor, 1989; Tanenbaum, 1990).

4. Relevant Issues

The author's experience in the subject lies in the following courses:

- *Digital Systems and Computer Architecture*, 3rd and 4th semesters on Systems and Informatics Engineering (since 1983): 3rd semester covers Combinatorial and Sequential Logic, Programmable Logic, and Microarchitecture and Microprogramming, as a case study of a particularly complex digital system and acting as bridge to **CA** ; 4th semester is for Computer Organisation, Machine Language (compiler and operating system interfaces), and Acceleration Mechanisms; Interfacing and Communications is dealt in a later course;
- *Computer Architecture*, 3rd semester on Maths and Computer Science (since 1988): the course contents is the same as the 4th semester in the engineering course;
- *Computing Systems*, 3rd and 4th semesters on Informatics on Management (starting 1992): the course contents during 3rd semester will be the same as the 4th semester in the engineering course, while the 4th semester deals with Interfacing and Communications.

Each one of these courses has exercises/laboratory classes. These are mainly used to teach the students assembly programming methodologies; the stress has been on methodologies rather than an in-depth analysis on a particular assembly language.

Some relevant issues are still open:
- what should come first: assembly programming or computer architecture?
- what is the best tool to introduce assembly programming?
- what type of programs should the students implement?

The teaching of assembly programming has followed a similar path to the teaching of HLL programming. In the 70's the use of a high level programming language was a course by itself, while in the 80's the students began to learn the concepts (algorithms and data structures) and later used a HLL as an implementation tool in the lab classes. In the 80's some students had assembly programming (of a given CPU) as a course by itself and, as a result, most of these graduates are tied to a specific CPU model and/or manufacturer and have difficulties to adapt to a new machine; current trend in some Universities is to emphasise the basic concepts on the instruction set design of a processor, and then use an assembly language as an implementation and evaluation tool in the lab classes.

The choice of an appropriate assembly language is not an easy task: commercial microprocessors are too complex to an introductory course, yet they represent a real architecture and the students are more motivated to them. A good compromise would be to use a simulator of a very simple architecture - like the one designed and implemented at Universidade do Minho, as a subset of i8086, or described in the literature elsewhere - which would introduce the basic concepts of assembly programming during the first weeks. The students then realise the limitations of such model and request a more powerful processor; by this time they are ready to practice with a real microprocessor, preferably with one that they can use in the labs to test the programs they implement (personal computers based on Intel microprocessors at Universidade do Minho).

Two main categories of exercises are usually available to the students to practice their skills in assembly programming: bit manipulation of peripheral ports for control-like applications, and compilation "by hand" of HLL programs. The first category may be useful for computer engineers, but the students do not learn the techniques of structured programming applied to an assembly language, and these exercises do not follow closely the lectures; the second type of exercises match quite closely the lectures and induce the students to specify the problem resolution through a HLL notation, stressing the quality and maintenance of the final product rather than speed.

As a concluding remark, it is feasible to successfully organise and lecture the same course on **CA** to students of Computer Engineering, Computer Sciences and Computer Business, provided a top-down approach is followed based on previous experience of the students on HLL programming.

References

Tanenbaum A.S. (1990): *Structured Computer Organisation (3rd Ed.).*- Prentice Hall

Tucker A.B. et al (1990): *Computing Curricula 1991.*- Report of the ACM/IEEE-CS Joint Curriculum Task Force.

Van de Goor A.J. (1989): *Computer Architecture and Design.*- Addison-Wesley.