HARDWARE CHALLENGES FOR COMPUTER SCIENCE STUDENTS

Nevena Ackovska, Milosh Stolikj, Sashko Ristov
Ss. Cyril and Methodius University / Faculty of Natural Sciences and Mathematic – Institute of Informatics
Skopje, Macedonia
e-mail: {nevena, milos, sasko}@ii.edu.mk

ABSTRACT

This paper describes the modifications made in the course Microprocessors and Microcontrollers, intended to challenge the computer science students to learn low-level hardware interfacing, interrupt handling, and other microprocessors issues, as well as embedded systems through learning microcontrollers. Implementing the changes lead to the improvements in many fields, such as better grade distribution, completion of many practical projects, increased number of passed students etc. It becomes more interesting, easy to learn, while still holding the necessary weight of an important academic course.

I. INTRODUCTION

Teaching computer science students how hardware devices work and how they can be employed in their designs is a very difficult process. Many different methodologies and approaches for teaching and learning computer science students about hardware from different points of view have been developed. In this paper we present our experience from teaching one such "hardware" course to students oriented towards software.

A. Background

Three hardware courses are obligatory at the Institute of Informatics. These courses are named "Computer Architecture" in the first study year, "Microprocessors and Microcontrollers" and "Modern Computer Systems" in the third. This paper focuses on the course "Microprocessors and Microcontrollers", its objections through the years, and changes made to the course in the direction to improve course syllabus and course bad reputation.

The course's main objective is for the students to obtain a clear understanding of issues such as low-level hardware interfacing, handling of interrupts, communication between processor, memory, bus and peripheral devices trough learning the basics of processors and its instruction set, as well as embedded systems through learning microcontrollers.

The course teaching is organized in three parts: theoretical lectures with 2 classes per week, theoretical exercises with 1 class per week and practical exercises with 2 classes per week in laboratory. Lectures and theoretical exercises are organized in larger groups, while practical exercises are carried out in computer laboratories in groups of up to 20 students, with each student working on its own workstation. Prerequisites for enrolling in the course are previously completed courses in Computer Architecture and Operating Systems.

During the semester, the course is divided in two parts. The first part covers the internal architecture and instruction set of x86 microprocessors, the interrupt handling system, BIOS and system routines. The second part focuses on various types of microcontrollers, analyzing their organization, instruction set and capabilities compared to x86 microprocessors, as well as peripheral systems, embedded systems etc.

Lectures and theoretical exercises for the first part of the course are based on [1] and various source code libraries. In previous years, programming was done using Microsoft Macro Assembler (MASM) version 6.11, with Programmers Work Bench (PWB) as an integrated development environment [2]. The material for the second part of the course depends on the microcontroller that was studied.

Grading in the course is divided into four categories - student activity, projects and either two midterms or one exam. Student activity is followed during the entire length of the course in the form of obligatory laboratory exercises, which are intended to be solved during classes. One project is handed out for programming in x86 assembly language. The projects consist of more complex problems which requires from the students additional research in the areas not covered in the course classroom. Finally, midterms, or later exams, are conducted for testing both theoretical and practical knowledge.

B. Course Objectives and Reputation

The course had a "bad" reputation of being abstract, boring and very hard to comprehend. Some of the issues were subjective and student-specific, while many others were completely in place. Most common objections to the course can be divided into three areas shown in the next sections.

1) Inappropriate programming and simulation environment.

Students had problems installing and using PWB, which caused aversion to the presented material and exercises.

2) Disjointed material between lectures, theoretical and practical exercises.

Students could not transfer knowledge gained from either lectures or theoretical exercises to practical exercises. This made them think they were studying two or three courses in one.

3) No "real world" application.

Without having direct hardware interaction, students learning becomes abstract, which leads to their displeasure and to the main question: Why we are learning this, and how and where shall I use it?

II. CHANGES IN THE COURSE

We analyzed the course during the period of several years, and noticed that there were many bad implications from the

course objectives previously mentioned. There were many students, who were national software contestants and winners, as well as CodeFu [7] Java contest contestants and winners, which had problems with passing the exams or the project, or were getting lower grades. Some of them were enrolled in the course more than once.

From this analysis, we decided to structure the course to resemble other "software" oriented courses. An approach like this enabled an easier starting point for our students since they were already familiar with high level programming languages like C++.

A. Changes in the laboratory exercises form

Most changes in the course originated from laboratory exercises. Totally new redesigned laboratory exercises were created and they were covering topics taught in the theoretical classes. Their structure was also modified in the form of tutorials. First they briefly cover the material given during theoretical lectures, including code samples how previously taught principles can be implemented. Afterwards, relatively simple assignments are given, requiring students to apply that knowledge for certain tasks.

The laboratory exercises difficulty increases during the semester.

B. Changes in the simulator for 8086 assembler

Initially we switched from using PWB for x86 assembler to modern visual simulators like emu8086 [3], where students can watch parameter values, i.e. registers, memory locations, as well as executing programs step by step in real time. They can use their computer science knowledge of debugging for easy troubleshooting. For the second part of the course, microcontrollers, we decided to switch from the previously used Intel 8051 architecture and PIC 8259A interrupt controller to some other implementation.

C. New Microcontroller platform – PIC 16F887

As a new platform we selected the microcontroller PIC16F887 [5], used on the EasyPIC6 Development System (Fig. 1) made by MikroElektronika [4].

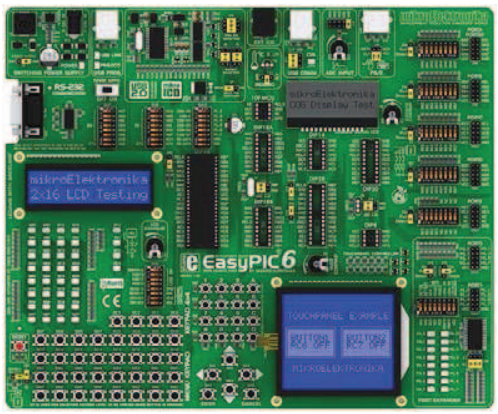

Figure 1: EasyPIC6 Development System.

We decided to use the PIC platform on the basis of the following premises:

1) Large number of implementations, including free and commercial.

2) The availability of the embedded software providing a direct connection with PC and the availability of Flash memory. This allows easy programming and re-programming of the memory as well as software adjustment and debugging.

3) A variety of controllers embedded in the microcontroller, which allows creating a lot of real world examples.

4) Free circulation of the IDE which allows reading, loading, adjustment and performance of software in a real-time mode and in a simulation mode.

5) Free circulation of a limited, but fully function version of the MikroC compiler for binaries up to 2KB.

6) Accompanying literature about the microcontroller and its programming [6].

7) The availability of many complete project examples for this microcontroller.

In specific, the PIC 16F887 microcontroller has RISC architecture and only 35 instructions. It can operate on frequencies up to 20 MHz, using an internal or external oscillator. Connection to peripherals is made through 35 programmable input/output pins. Three types of memory are available – 8KB ROM memory, 256 bytes EEPROM memory and 368 bytes RAM memory. In combination with the EasyPIC6 Development System the powerful In-circuit Debugger enables real time execution and debugging of code directly on the microcontroller, with options for step by step execution, monitoring of memory, registers etc.

The company MikroElektronika offers variety of hardware (from PIC series), as well as a lot of software support. This enabled us to include a lot of sensors, actuators, as well as small control systems as part of exercises and exams.

D. Changes in the laboratory exercises content (First Part)

For the first part of the course, we created totally new content of the lab exercises. We omitted the old "hardware" oriented exercises which taught the students working with graphics, operating systems, files etc, but introduced more software-like exercises such as arrays and matrices, strings, procedures, macros etc. The exercises were organized as follows:

1) Introduction with environment and simulators

Students were introduced to the programming environment, the Microprocessor Simulator Emu8086, used in virtual machine environment. The students learned how traffic lights on simple crossroads can be programmed with few lines of code. After that, they proposed how traffic lights can become more intelligent.

2) Introduction with assembler for 8086

Students were introduced with the basics of 8086 processor design, different types of registers and basic instructions like MOV, ADD, SUB, CMP, and LOOP.

3) Comparison with high level languages

Students learned different ways of addressing and accessing the memory. Also, they learned how to program high level commands like IF-THEN, IF-THEN-ELSE, WHILE, FOR, and REPEAT UNTIL.

4) Introduction with arrays and matrix

Students learned how to define arrays and matrices in the linear segments and store them in memory, as well as access their elements with different sizes.

5) Working with strings

Students learned memory representation of strings, as well as the instructions for string management.

6) Procedures, Macros and Interrupts

Students learned advanced methods of programming with procedures and macros, as well as programming interrupts, especially INT21.

E. Changes in the laboratory exercises content (Second Part)

For the second part of the course, we also created totally new content of the lab exercises, according the new microcontroller and its capabilities with the development tools. The exercises were organized as follows:

1) PIC16F887 Working Environment

Students were introduced with the working environment of the microcontroller, the programming language microC, as well as executing basic programs such as led blinking in the simulator.

2) Counters and Timers in PIC16F887

Students were introduced to counters and timers of the working environment, their usage and programming, upgrading the previous exercise.

3) Serial Communication in PIC16F887

Students were introduced to the EUSART module for serial communication. They programmed the controller to read and write data with the computer serial port, sending bytes, strings, or integers.

4) LCD, KeyPad and EEPROM in PIC16F887

Students were introduced to an additional hardware component – LCD display. They also learned to use additional memory in order to handle memory deficiency.

5) Advanced and real problems

At last, students solved some real issues using all previously learned hardware components into one exercise.

F. Changes in the obligatory projects

Also, changes were made in the practical projects. In previous years, many of the students did not even start the projects and did not complete them at all. To improve this, we split the material for the projects into two parts, according to midterms. For instance, students which passed the first midterm are assigned a project from the material from the second midterm, and opposite, those who did not pass the first midterm, must learn the first midterm material. Therefore, they are assigned a project from the first midterm.

Projects from the first midterm are connected with the previous obligatory course – Computer Architecture. Some examples projects are: number system conversions into a given file, implementing calculator, CRC code, etc.

Projects from the second midterm are of similar weight, but intended for the PIC microcontroller framework. Some examples of such projects are: working with serial port, implementation of timer and counter, working and manipulating with arrays, implementing alarm system, etc. All these projects are suppose to be programmed into previously learned simulators on hand-on labs.

G. Introducing an optional practical project

At the beginning, we offered 10 optional practical projects from the real world, such as popular games or controlling some well known processes. All these projects must been mandatory performed on the hardware components, which weren't previously studied.

III. RESULTS

The results from the course changes can be observed from several aspects. One very obvious is the improvement in grade distribution. The number of passed students from the midterms and the first exam increased to 28%, and the average grade was 0.24 greater than previous year. The number of the students which submitted the projects on time increased 7%, as well as almost 60% performed better result in the laboratory exercises, compared to the previous year.

IV. CONCLUSION

From the results shown, we believe that changing the syllabus; we challenged the students to learn hardware, using their software based skills. They solved the basic hardware limitations of the memory usage.

We believe that these changes will be incorporating not only in to hardware courses, but also to all other courses where these changes and improvements are applicable.

REFERENCES

[1] Randall Hyde, "The Art of Assembly Language", ISBN 1-886411-97-2

[2] Microsoft Macro Assembler, http://www.masm32.com/history.htm; Retrieved on 02.02.2011

[3] Emu 8086, http://www.emu8086.com; Retrieved on 02.02.2011

[4] Mikroelektronika, http://www.mikroe.com; Retrieved on 02.02.2011

[5] Microchip PIC16F887, http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en026561; Retrieved on 02.02.2011

[6] microC PRO for PIC, http://www.mikroe.com/eng/products/view/7/mikroc-pro-for-pic/, Retrieved on 02.02.2011

[7] CodeFu coding competition. http://codefu.mk/ [02/15/2011]