The Training Environment for the course on Microprocessor Systems at the Politecnico di Torino

M. Rebaudengo, M. Sonza Reorda[1]
Politecnico di Torino
Dipartimento di Automatica e Informatica
Torino, Italy

Abstract

This paper describes the training environment used for the course on Microprocessor Systems held within the curriculum toward the MS degree in Computer Science at the Politecnico di Torino. The environment is composed of inexpensive hardware and software material that allows students to perform a final assignment work based on implementing a stand-alone video-game system. The characteristics of the hardware and the specifications of five games are described. The experience gathered in the past five years is referred to in explaining the advantages of this approach: more effective learning of microprocessor system architecture and programming, practical experience in project management, first contact with complex systems handling.

1. Introduction

This paper presents the training environment (hardware and software material, exercise specifications) used in the undergraduate course on Microprocessor Systems held at the Politecnico di Torino.

The course presents the basics in Microprocessor Systems architecture and programming, and adopts as a reference the Intel 8086 family [1]; it overviews the architecture and assembly language of the processor, as well as the use of the family's main peripherals (serial and parallel interface, programmable timer, interrupt controller). The course is located in the middle of the curriculum leading to the MS degree in Computer Science: previous courses teach the students about the basics of Computer Science, high-level languages programming, algorithms and data structures; further courses provide knowledge about more advanced microprocessors and microcontrollers architectures, as well as Operating Systems and Software Engineering. This course is thus designed to provide a basic knowledge about microprocessor systems, and to give the students a real experience in software programming of simple boards based on a microprocessor and related peripherals.

The Intel 8086 family has been chosen for two reasons: first, the large availability of Personal Computers for experimenting the Assembly language, and second, the requirements of several local industries working in the PC market and interested in recruiting engineers able to write drivers for specific devices and boards.

The laboratory exercises consist of two parts: the first comprises a set of guided exercises focused on single topics (language constructs, peripheral programming, etc.); the second calls for a final assignment corresponding to a project students must carry out. Since training in microprocessor board programming is the prime aim, this assignment requires the programming of an existing hardware. The definition of the hardware and software environment, as well as the specification for these assignments were strongly influenced by the following requirements:

- the size of the assignment should be large enough to simulate a real project, but small enough to be completed within the course;
- its specifications should cover the maximum number of course topics to ensure their fuller acquisition through practical experience;
- they should be challenging and interesting enough to strongly motivate students during the execution of their assignment;
- they should be vague enough to allow the choice of a variety of solutions, but precise enough to allow comparison of the assignments handed in;
- hardware and software should not be expensive;
- the hardware should be flexible to enable it to be employed for future assignments and thus achieve a good ROI (*Return On Investment*).

These requirements were met by providing the students with all the hardware needed to produce a simple stand-alone video-game system and asking them to write the software implementing the game. The experimental results we gathered by adopting this approach in the past five years have demonstrated that it fulfills most of the goals.

The paper focuses first on the hardware material (Section 2), and then on the specifications of the work to be done by the students (Section 3). Some examples

[1] Contact address: Matteo SONZA REORDA, Politecnico di Torino, Dip. Automatica e Informatica, Corso Duca degli Abruzzi 24, I-10129 – Torino, Italy, Tel. + 39 11 564 7055, Fax + 39 11 564 7099, E-mail sonza@polito.it, http://www.polito.it/~sonza

are given in Section 4, and the results are assessed in Section 5. Conclusions are drawn in Section 6.

2. Hardware Material

The hardware material is composed of 3 parts (Fig. 1): a PC, a microprocessor board and a peripheral device.

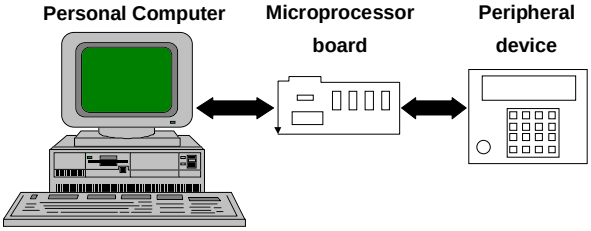

Fig. 1: The training environment.

2.1 Personal Computer

This provides a simple and well-known development environment. It is employed to write, assembly, and download the Assembly code running on the microprocessor board. Any kind of PC (from 8086 PC on) is acceptable.

2.2 The Flight86 board

The microprocessor board is a commercial board called Flight86 [2] designed and sold by Flight Electronics Ltd.

It is designed to simplify teaching of the 8086 CPU and some of its commonly used peripherals, and is equipped with a 8086 microprocessor, 16 Kbytes of RAM, 16 Kbytes of ROM, 2 parallel interfaces (8255), a serial interface (8251), an interval timer (8253) and an interrupt controller (8259). A serial port connects the board to a PC through the RS-232 protocol; hardware interfacing of peripheral devices is achieved through two parallel I/O ports.

Two programs are provided with the board: a Monitor program resident in EPROM, and the host software for the PC. The main operations allowed by the software are code downloading from the PC onto the memory board, program running and debugging. Several facilities permit an effective debug, such as inserting breakpoints into the code, running the program step by step, controlling the values stored in the registers and in the memory, and disassembling the code. Once the code has been debugged and the final version downloaded into the board memory, the Flight86 can be isolated from the PC and act as a stand-alone machine.

2.3 The Programmable Video-Game Device

The Flight86 needs some additional device to interface with the outside world; for this reason we designed a peripheral called the *Programmable Video Game Device* (*PVGD*) formed of 3 parts assembled into a single box: a Liquid Crystal Display (LCD), a keyboard and a buzzer. It is connected to the Flight86 through a connector.

The LCD [3] has an alphanumeric display of 160 characters (4 lines of 40 characters): each character is formed of 8x5 dots. The LCD contains a built-in Hitachi HD44780 controller chip that allows it to receive data directly from a 4-bit or a 8-bit microprocessor or microcontroller. The chip has 192 character patterns in a ROM memory. A programmable RAM memory is also available, allowing the user to define up to 8 new characters.

Data for display on the module are sent through the data bus from the microprocessor to the LCD controller via a parallel interface. Data transmission is programmable and can be performed in either 8-bit words or 4-bit nibbles. The input lines to the LCD controller are two enable signals, a read/write control signal, a control register signal and 8-bit data bus lines.

The keyboard has 16 keys organized in 4 rows and 4 columns. Row and column lines are connected to an output and input 4-bit port of the Flight86 board respectively.

The buzzer is a piezo-electric element that transforms alternating voltage into acoustic signals and can be driven by sinusoidal or square waves.

The interface between the Flight86 and the PVGD is designed to use the three ports of one of the two 8255s and one counter of the 8253 to drive the PVGD peripherals.

2.4 Cost

The total cost for the hardware is about $600 for the Flight86, including the software, and about $200 for the PVGD, including material, assembly, and testing costs.

A PC equipped with a text editor and an Assembler program is available for each student. Their cost is not included because they are not specific for the course and also used for other courses.

The total cost is regarded as acceptable, since we expect the system to be in use for at least 8 years and support the execution of a wide set of assignments.

3. Work Specifications

As already stated, students are required to program the hardware and produce a stand-alone video-game system composed of the Flight86 board driving the PVGD.

Video-games were chosen for several reasons: first of all, their characteristics are normally well known, which makes it easier to define the specifications and reduces the risk of misunderstandings; secondly, despite its relatively low complexity, construction of a video-game raises several interesting problems, such as graphical and sound interface definition and implementation, real-time programming, and concurrent processes coordination; lastly, students are much more motivated in implementing a game than other applications. Some basic notions about game theory and game programming [4] are provided before they start their assignment.

Students are organized in groups, each composed of three persons. Each group can access the laboratories where the hardware and software material is available during a period of about two months without any restrictions. Each student group is also allowed to loan the Flight86 board and the PVGD, so that they can work at home, if they own a PC. After a specified period, the groups are asked to deliver their final product in the form of a floppy disk containing the source code of the program and some related documentation corresponding to its *Installation Guide*, *User Guide*, and *Product Reference Manual*.

Students are provided with very simple specifications based on well-known and simple games (see below for details) that also allow them to enrich what they produce according to their own fancy and capability.

The following parameters (in order of decreasing importance) are used to evaluate the work of each group:

- conformity to specifications
- user-friendliness of the interface
- completeness of documentation
- robustness, modularity, and efficiency of the implementation
- improvements.

4. Assignments

We will now describe the works assigned for implementation in recent years to show what the hardware described can be exploited for. Further specifications could readily be devised for the same hardware. Works assigned are always a simplified version of well-known video-games:

- *Hunter*: a target runs in a random way on the screen. The player can move the hunter via the keyboard, rotate his gun towards the target, and shoot. If the target reaches the hunter before being hit, the hunter dies.
- *Squash*: a single player moves a racket around the court and hits a ball, which bounces off the walls bounding the court along three sides. A maximum number of balls are allowed to leave the court through the back side. The score corresponds to the number of rebounds against the wall.
- *Tetris*: Pieces with simple shapes and composed of a small number of square blocks fall one after the other from the top of the screen. The player may rotate each piece and move it horizontally as it falls, to make it fit in with those that pile up at the bottom of the screen. When a solid row of blocks (no holes between the pieces) forms in the bottom, the row disappears. Otherwise, the pieces continue to pile up and reduce the space to be left for new pieces, thus increasing the difficulty of the game.
- *Grand Prix*: the screen represents a car racingtrack, like the Indy one. The player moves a car via the keyboard. The score corresponds to the number of laps covered by the player in a fixed amount of time. The games is over if the car goes off the track or the time terminates
- *Worm*: the player moves a worm across the screen to collect as many apples among the ones randomly apperaing on the screen. The longer the worm lasts and the more apples it eats, the higher is the score. The game is over if the worm crashes into a wall or into its body.

4.1 Characteristics

An important feature of any implementation of these games is that the program is composed of several modules performing different tasks. A *Main Module* orchestrates the other modules and runs the core algorithm. A *Timing Process* provides the time information required to coordinate the execution of the other modules: a polling management could be used, but better results can be obtained with an interrupt mechanism to activate each procedure. A *Graphic Interface* displays the characters on the LCD; the displaying of objects in motion and the definition of new characters greatly increase the number of features which can be implemented on the LCD. A *Music Interface* drives the buzzer and could be made very sophisticated to produce a musical background. A *Keyboard Interface* manages the keyboard: difficulties arise from the correct management of the arrow keys, which in an action game corresponds to a real-time problem.

The complexity of the task (the final code normally amounts to some thousands of lines) means that students must carefully organize their work: they first identify the high-level modules, and then proceed in a top-down approach that can sometimes be combined with a bottom-up approach, by introducing a procedure library to manage specific hardware components (e.g., the LCD, or the buzzer). Some parts of the code (e.g.,

the one implementing the core algorithm) can be written in C, provided a suitable interface towards the other modules is defined. Modularity also eases the task of sharing the work among the components of each group.

5. Evaluation of the Results

Implementation of a video-game machine using the material described was first introduced five years ago. A different game was assigned each year and data were collected on the number of groups to which the work was assigned, of the groups that delivered a successful product, and of the groups that added some new features. As can be seen in Tab. 1, about 75% of the groups were able to deliver a working product, demonstrating that the main goal of the course, i.e., providing the students with the ability to solve a real problem on a microprocessor board, had been reached by an acceptable percentage. About 50% of the groups introduced additional features: these mainly concerned new game options (e.g., transforming *tetris* game into a 2 players game), sophisticated graphic interfaces, and musical effects.

The didactic advantages of adopting this environment for implementation of the games will now be described.

Game	*Groups*	*Successfull*		*With new Features*	
	#	#	%	#	%
Hunter	27	21	77	14	52
Squash	30	23	76	17	56
Tetris	32	24	75	18	53
Grand prix	31	23	74	16	51
Worm	33	25	75	18	54

Tab. 1: Group activity.

5.1 In depth knowledge about Computer Architecture and Assembly Language

The work assigned requires about 2 full-time weeks per student and a high degree of involvement. It thus forces students to acquire a very good knowledge of both the 8086 assembly language, and the architecture and characteristics of a 8086-based board; we noted significant improvements in this connection compared with the conventional laboratory exercises (previously adopted for the same course); this result has been reached without a significant increase in the time required by the students to prepare their final examination.

5.2 First Expertise in Advanced Topics

The assignments require the implementation of a set of concurrent processes: one for keyboard handling, another for updating the LCD, an optional one for playing sounds, and one for computation. The processes can be managed either with a polling strategy or by resorting to the interrupt mechanism. They possibly share data, and some of them must be accessed in a mutual-exclusive way. Time constraints have to be considered and solved to devise a correctly working product. A number of important problems are thus present, and students deal with them first from a practical point of view on simple cases. Further courses (e.g., the one on *Operating Systems*) provide them a more complete and theoretical framework, and a set of complete solutions. We believe that the illustration of practical problems, followed by the provision of solutions, is well-suited for a technical university. It has certainly been very well accepted by the students.

5.3 Real Experience in Project Management

The complexity of the software, which must be structured in several modules (possibly corresponding to processes), requires students to devise a logic architecture for their program. This planning phase must be based on a preliminary evaluation of the advantages and disadvantages of the possible solutions.

Moreover, the specifications we provide are generally quite vague, especially with regard to the user-interface and the features that could be added to the basic game: this forces students to devise solutions and to design alternatives. More importantly, as they are also in charge of putting their ideas into practice, they must evaluate how expensive and risky (in terms of robustness) their solutions could be. Our experience suggests that most groups that fail do so because they add excessively complex features to the basic specifications. The main weakness here is not a lack of implementation capabilities, but an inability to evaluate the complexity of what is going to be implemented.

Completion of an assignment thus constitutes a real experience in project management, something that has been previously indicated as one of the most serious weaknesses of our students, and which will be more deeply dealt with by the following course on *Software Engineering.*

6. Conclusions

We have described the training environment used for the course on microprocessor systems held at the Politecnico di Torino.

The environment is based on a commercial board equipped with a 8086 microprocessor and several peripheral devices from the same family; the board is connected to an ad hoc designed peripheral device containing a keyboard, an LCD, and a buzzer. Programs to be run on the system can be edited and compiled on a PC, and then down-loaded and debugged on the system itself.

Students exploit the described hardware to develop a real product corresponding to a stand-alone video-game system. Our experience over the past five years demonstrates that this approach leads to the following important results:

- thanks to their great involvement, students get a deep knowledge in microprocessor systems architecture and programming;
- they experiment how to manage a real project and understand where some critical problems must be faced, e.g., in the definition of a suitable user interface;
- they work on a system complex enough to acquire an initial experience of advanced topics they will deeply deal with in further courses, such as *Operating Systems* and *Software Engineering*.

The best works done by the student groups for this course in the past years are available at the URL http://www.polito.it/Ulisse/CORSI/INF/N0460/material e/tesine/.

7. References

[1] Yu-Cheng Liu, G.A. Gibson: *Microcomputer Systems: the 8086/8088 Family*, Prentice-Hall, 1986

[2] *Flight86 Microprocessor Training Equipment: Technical Reference Manual*, Flight Electronics International Ltd., Southampton, UK, 1991

[3] Liquid Crystal Displays Alphanumeric and Graphic Modules Standard and Custom Design, Varitronix Ltd., Hong Kong, 1991

[4] E. Solomon: *Games Programming*, Cambridge University Press, Cambridge, UK, 1984