Int. J. Engng Ed. Vol. 23, No. 4, pp. 785–798, 2007
Printed in Great Britain.

0949-149X/91 $3.00+0.00
© 2007 TEMPUS Publications.

An Integrated Microcontroller-based Tutoring System for Computer Architecture Laboratory Course*

D. E. BOLANAKIS,[1] E. GLAVAS,[1] G. A. EVANGELAKIS[2]
[1]*Department of Communications, Informatics and Management, Epirus Educational Institute of Technology, Arta, 47100 GREECE. E-mail: dmpolanakis@teleinfom.teiep.gr and eglavas@teiep.gr*
[2]*Department of Physics, University of Ioannina, Ioannina, 45110 GREECE. E-mail: gevagel@cc.uoi.gr*

This paper presents the framework of a microcontroller-based approach for a computer architecture laboratory course intending to reinforce the educational level of non-electrical/electronic students, placing the emphasis on hardware and software design issues for embedded computers. The difficulties encountered in this approach are related to a) focusing students' attention on efficient use of the laboratory's equipment instead of tutoring, a common risk in technological courses, b) overcoming the comprehension barriers that are a consequence of the educational level of the students. The former issue is addressed by the design of appropriate equipment for the laboratory, the latter by a pedagogical strategy that is based on representational and interpretational picture examples. The proposed methodology proved to help in bridging the gap between the design problem and the students' semi-formal design view, without missing the essential details of the tutoring.

Keywords: computer architecture; microprogramming; microcontrollers; system design; illustrations.

INTRODUCTION

TEACHING MICROCONTROLLERS has been for many years a necessity and a challenge. Besides their broad usefulness, microcontrollers can also serve as an effective method of initiation into a microcomputer/microprogramming laboratory course. Therefore, microcontrollers have been installed in many microcomputer-based laboratories (MBLs), to provide an important tool in the education of students in industrial electronics [1] and instrumentation and measurement [2]. Accordingly, one might wonder whether a microcontroller could be used in a computer architecture laboratory course and with what benefits and drawbacks. An anticipated answer is that since a microcontroller constitutes a complete computer system, it could be so used. Indeed, introducing a complete computer system to a class, albeit on a single chip of limited abilities, would keep students focused on the basic components of a computer's architecture, without confusing them with the plethora of detail in complex computer systems. In addition, firmware development for the microcomputer would enrich their backgrounds with hardware/software design issues for embedded computer systems. Unfortunately, in a curriculum that is mainly specialized in topics other than electronic engineering, the selection of microcontrollers for laboratory training in computer architecture entails the risk of confusing students with an onslaught of new information. To avoid this, the present communication focuses on the perception difficulties of perhaps inadequate students and proposes a training framework that can be summarized as follows:

- low-level programming concepts;
- microcomputer architecture and microprogramming;
- system design consideration for specific microcontroller-based applications.

The two-hours per week laboratory course, takes place in the second academic year of the syllabus, at the Department of Communications, Informatics and Management, Epirus Educational Institute of Technology, Arta, Greece. Curriculum content is focused mainly on informatics. At this level of education, the students are already familiar with digital design concepts, from the prerequisite course in 'digital electronics' and with high-level programming languages. Under these conditions, when teaching a microcontroller-based course, there are two main issues:

a) the pedagogical strategy;
b) the selection of equipment.

Difficulties with the first topic are related to firmware development using low-level programming techniques as well as establishing a clear link between the firmware and hardware. Problems with the second topic are related to the risk,

* Accepted 16 January 2007

common in technological courses, of drawing students' attention away from the goal of the course, by focusing on too much technical detail.

The abundant literature sets out three different approaches. The first one relies on education through simulations [3–6]. The advantage of this method is that it is low cost and can be easily implemented in the laboratory. However, it suffers from the limited ability of simulators to describe technical problems and details that appear in applications, e.g. the appearance of the bouncing phenomenon with mechanical switches, or the illumination modifications of a lamp dimmer resulting from a pulse width modulation. Such limitations prompted educators to develop customized simulators, mostly focusing on the simulation of embedded/external peripherals [3, 4].

The second approach focuses on design and implementation of the units that constitute a computer [7–9]. The advantage of this approach is the experience students gain in digital design and implementation, as well as obtaining a good general knowledge of the computer's components. Nevertheless, there is almost no programming with this approach, resulting in the impression that hardware and software development are two separate issues.

The last approach uses commercial or custom training kits in which the students develop firmware for the microcomputer. Many researchers are convinced that use of simulations alone does not provide sufficient illustration of actual applications [10], compared with real hardware which gives students a sense of accomplishment and they work harder [11]. However, there are some drawbacks:

a) building up a customized microcomputer/circuit is a time-consuming procedure which leaves no alternative but either to design and implement the system at home [12] or to conduct the experiments over the internet [13];
b) adopting a commercial development kit which requires the course to be adjusted to the features of the kit, thus reducing its educational efficiency. In addition, in most commercial boards some wiring has also to be done during the course, a time-consuming procedure even for three-hour courses [14].

Educators who take on the teaching of microcontrollers to non-electrical/electronics engineers choose one of the above approaches, and propose methodologies to surmount the barriers of understanding. A remarkable effort that is in line with the use of kits [15] consists in building the microcontroller course around real-world applications. This method meets ABET's [16] requirements, uses the microcontroller as a design tool to solve a problem, and to avoid students becoming confused, abstains from architecture details and low-level programming concepts. Although this approach has been proved to be very motivational for the students, it is not, however, suitable for a computer architecture laboratory course. Details of the microcontroller as well as assembly language learning are essential to improve course education. The consequential question is 'how can someone bridge the gap between the design problem and the students' semi-formal design view, without missing out essential details of the tutoring'?

A promising answer was revealed in Levin's suggestions [17]. Perception difficulties have been addressed with the use of representational and interpretational pictures, which are proved to aid understanding [18] and enhance learning [19]. Besides the pedagogical strategy, this effort has the drawbacks of the selected approach and requires a customized training kit with the following specifications:

- explicit geometry for quick and easy use;
- minimum work on hardware implementation, so the board can be ready for use during the lessons;
- maximum number of practical examples, to provide experience in microcontroller-based applications;
- expandability.

Pedagogical strategy and the design of efficient equipment are the basis of a computer architecture course that is particularly relevant for students who are not thoroughly immersed in a specific area.

HARDWARE

Figure 1 is a schematic diagram of the educational learning board. The main component is the 8-bit microcontroller unit (MCU) MC68HC908GP32. The board is designed to support voltage regulation and protection against power inverses and short circuits. It also supports in-circuit simulation/debugging and programming of the microcontroller's flash memory through a RS232 serial link (DB9-A). The configuration mode is enabled through six slide switches (SW-PTC0, SW-PTC1, SW-PTC3, SW-IRQ, SW-PTA0 and SW-PTA7). To avoid mistakes during configuration, the programming mode can be selected by setting the switches in the same direction.

Peripherals attached to the educational board, are the most popular devices commonly used in microcontroller applications. A red and a green colour light emitter diode (LED1 and LED2) are connected to the microcontroller's lines PTD4 and PTD5. The LEDs can also be forced by the timer interface module (TIM), a shared function with the simple I/O pins. Two pushbuttons (SW1 & SW2) are connected to the PTC0 and PTC1 lines, while a third pushbutton (SW3) is connected to the external interrupt line (IRQ). Two seven segment displays (TENS and UNITS) are forced by the same I/O lines (PTB [6:0]) and are activated/deactivated through PTD4 and PTD5 lines. A 16-key matrix-output keyboard (MATRIX) is

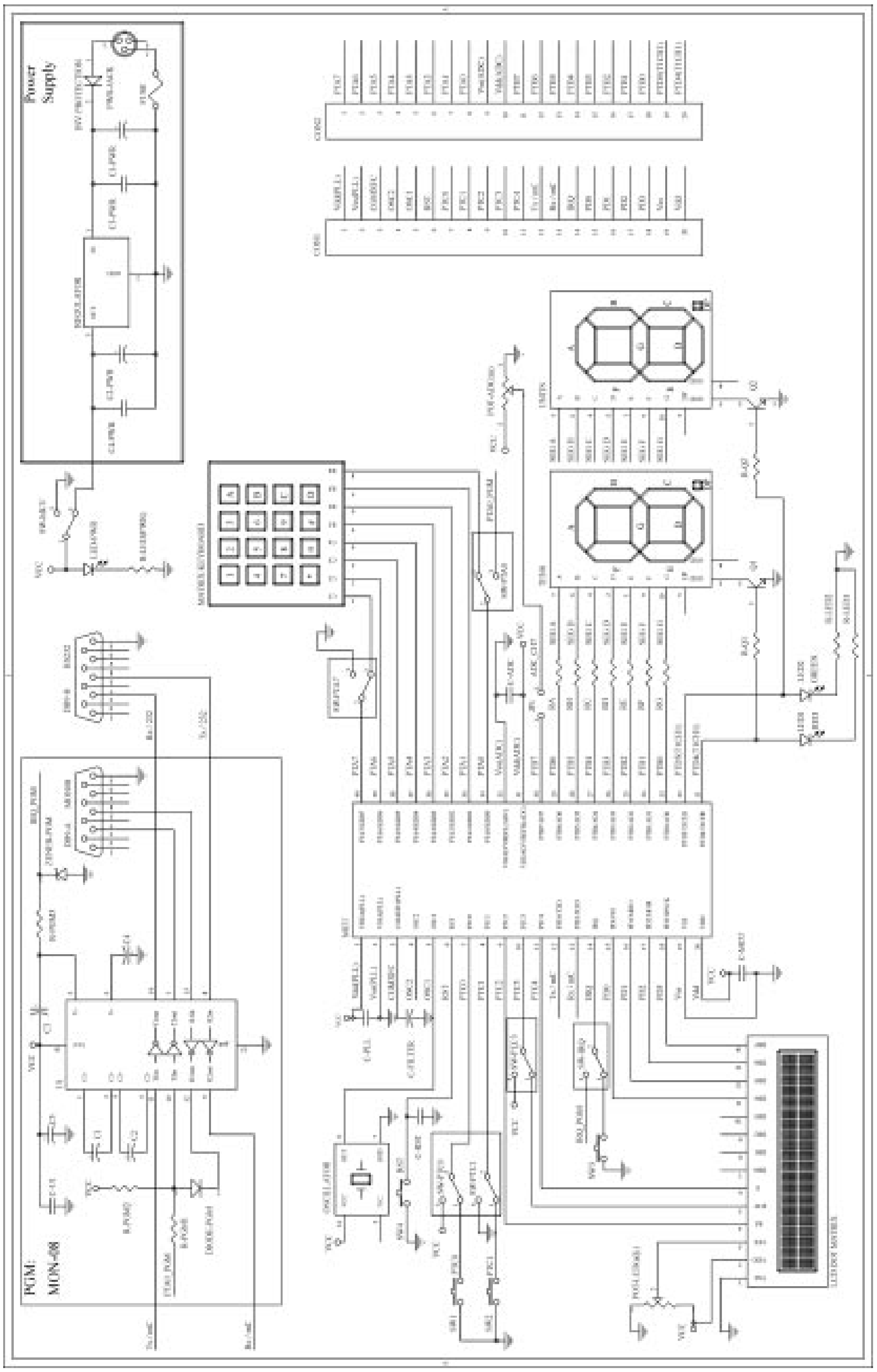


Fig. 1. Schematic diagram of the educational learning board (ELB)

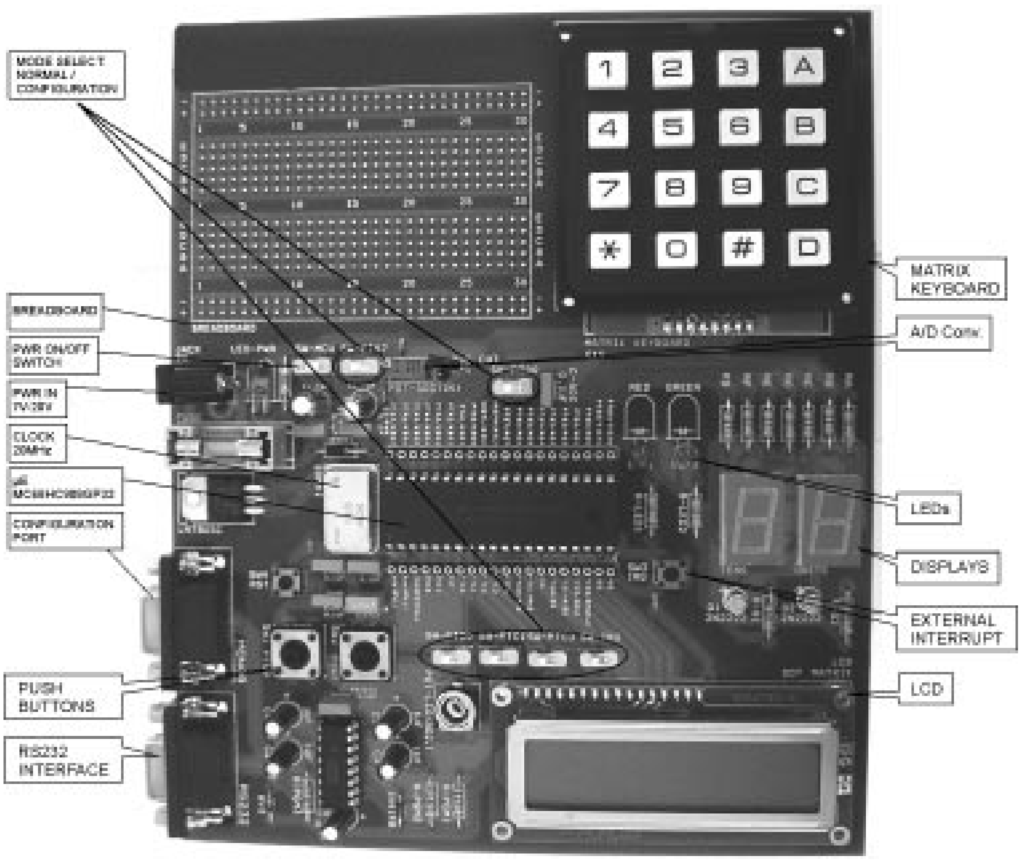


Fig. 2. Photo of the educational learning board (ELB)

connected to the lines PTA [7:0]. An RS232 serial link (DB9-B) is used with the serial communication interface (SCI) subsystem on the microcontroller (lines PTE0/TxD, PTE1/RxD). A liquid crystal display (LCD DOT MATRIX) is connected to the lines PD[3:0] for a 4-bit bus access, and is controlled through the lines PTC2, PTC3 and PTC4. Finally, a trimming potentiometer (POT-ADCin) is connected though a jumper (JP1) to the PTB7 line, as an input to the analogue-to-digital (ADC) subsystem.

Figure 2 shows the development kit of the educational learning board. Its dimensions are 16 cm $\times$ 19 cm. For low cost reasons, the board is designed in a single layer. The silk screen on top of the board has a detailed description of the materials. Adjacent to the microcontroller are two sockets (CON1, CON2) which are connected to the microcontroller's pins. The names of these pins are noted on the silk screen at the side of the connectors. This facility, along with a small breadboard in the upper left area of the kit, makes this expandable learning board useful for students whose thesis requires them to design custom circuits.

PLANNING THE COURSE

The course outline (Table 1) is separated into two basic sections. The first section (labs 1–4) includes an introduction to the microcontroller's fundamentals and to the mnemonic source code development (structure of the central processor unit, registers, memory organization, vectors, clock cycles, assembler's syntax rules, directives, etc), as well as a familiarization with the laboratory's software and the hardware equipment (the simulator's user interface environment and the educational learning board). The second section (labs 5–13) is dedicated to practical examples, where flowcharts are used as planning tools for the code development, the assembly language instructions are selected to accomplish the program code and the program code is simulated (and/or in-circuit simulated) and verified for the microcontroller.

In the first part of the course, we use the representational pictures approach, introduced by J. R Levin in 1981. In the second part, there are two critical points:

a) the occupation of students on hardware implementation is time-consuming and might distract them from acquiring knowledge about the architecture and handling of the computer system;
b) it is not trivial to make a clear link between firmware and hardware.

To overcome the former issue, an educational learning board was designed, taking into account

Table 1. Course outline

Labs	Topics	Objectives
Lab 1	Introduction	• Man and machine communication • Introduction to microcomputers • CPU08 & M68HC908GP32 architecture
Lab 2	Assembly language	• Low-level programming concepts • Microprogramming • Instruction set summary
Lab 3	Simulation	• Familiarization with the simulator • Simulation instructions & tips
Lab 4	Hardware (design details)	• Hardware/software design issues • Familiarization with the educational learning board • In circuit simulation / debugging
Lab 5	Status LEDs	• I/O peripherals • Data output to the outside world • Transistor switch example • Shared lines considerations • Delay techniques
Lab 6	Push-buttons	• I/O peripherals • Data input to the microcontroller • Bouncing effect • Hardware debouncing techniques • Debouncing delay
Lab 7	7-Segment display	• I/O peripherals • Data storage to flash memory • Indexing techniques • Multiplexing techniques
Lab 8	Matrix keyboard	• I/O peripherals • Matrix keyboard interfacing algorithm
Lab 9	Liquid Crytal Display	• I/O peripherals • Timing diagrams & constraints
Lab 10	Serial Communication Interface	• Embedded peripherals • Serial communication (asynchronous) • Alphanumeric communication • Numerical conversions
Lab 11	Pulse Width Modulation	• Embedded peripherals • Timers; input capture, ouput compare, PWM
Lab 12	Analog to Digital Converter	• Embedded peripherals • Signal acquisition • Numerical operations
Lab 13	External Interrupt	• Embedded peripherals • Interrupt handling & priorities

the needs of the course. The aim was to provide the opportunity for practicing several microcontroller applications, focusing on firmware development and study of circuit behaviour under real conditions. Taking into account the time constraints of the course, the board was designed with no requirements in hardware implementation. The latter issue was addressed by the concept of inter-

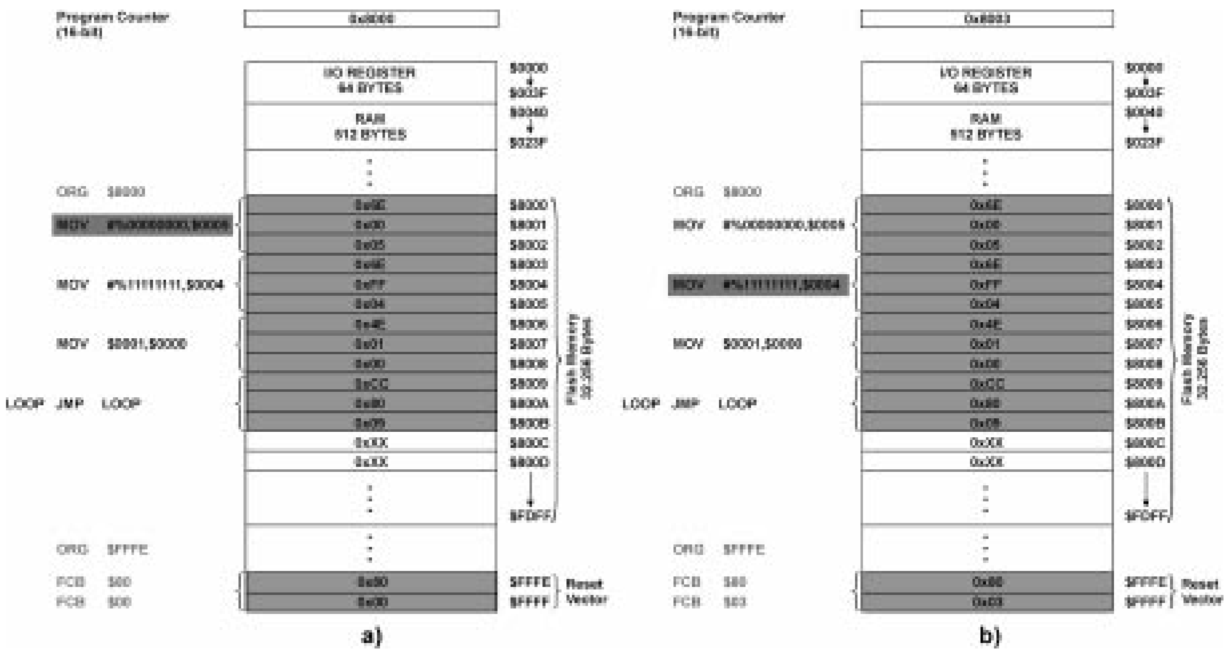


Fig. 3. Representational pictures: mnemonic source code and machine language

pretational pictures, as introduced by J. R. Levin. The interconnection between firmware and hardware depicted in parts-and-steps [20] picture examples, proved able to tackle comprehension difficulties.

First part of the course: representational pictures

The representational example of Figure 3 describes three actions for:

a) loading executable code into the microcontroller's memory and initiation of execution;
b) stalling of the code execution;
c) establishing the interface between the microcontroller and the outside world.

Action 1: initiate the code execution

Every program must be loaded into the computer's memory before execution can begin. In high-level code development, this action is carried out by the linker. The linker inserts the missing sets of instructions into appropriate places to form a file, and attaches a loader to the start of the executable file. In low-level code development, the software designer uses the appropriate assembler directives to assign the executable file to the microcontroller's memory and attaches a loader to the program counter (16-bit register that contains the address of the next instruction to be fetched). Additionally, the designer must be conscious of the memory locations that are disposed for the executable code assignment in the microcontroller unit. As far as these actions are not a matter of high-level code development, the students are confused with their use in the assembly language code.

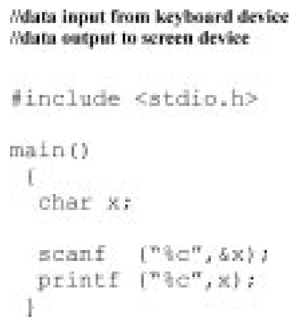
Figure 3 a) and b) depict the assembly language of a simple source code and its correspondence in machine language assigned in the microcontroller's flash memory. The memory locations in grey contain the machine language instructions and data, while the rest are unimplemented (signed as 0xXX). The brackets in the pictures define the machine language instructions from the corresponding original. The assembler's directives for the source code are noted in grey type, while the instructions are in black. It is quite clear that the pseudo-instruction ORG $8000 is used to assign the program code to the very beginning of the flash memory, while the last three directives assign values to the reset vector located at $FFFE and $FFFF. At the same time, the meaning of the pseudo-instruction term becomes clear, since the directives, normally present in assembly code, are now missing from the flash memory. During reset, the program counter is loaded with the contents of the reset vector. The incorrect assignment of the reset vector in Figure 3 b) causes failure of the execution of the first instruction of the code.

Action 2: stall the code execution

Unlike PCs, the microcomputer unit does not automatically break the execution of the code after the completion of a task. The designer should take advantage of flow-control statements to perform similar actions. Looking at the machine language in the figure is more concrete for the students to understand how to stall the sequential execution of the code, by perpetually executing the unconditional JMP (JuMP) instruction. This sends the code's execution to the program's memory location, labelled LOOP.

Action 3: establish *the interface between the microcontroller and the outside world*

Programming the microcontroller presupposes knowledge of specific memory locations, i.e. I/O



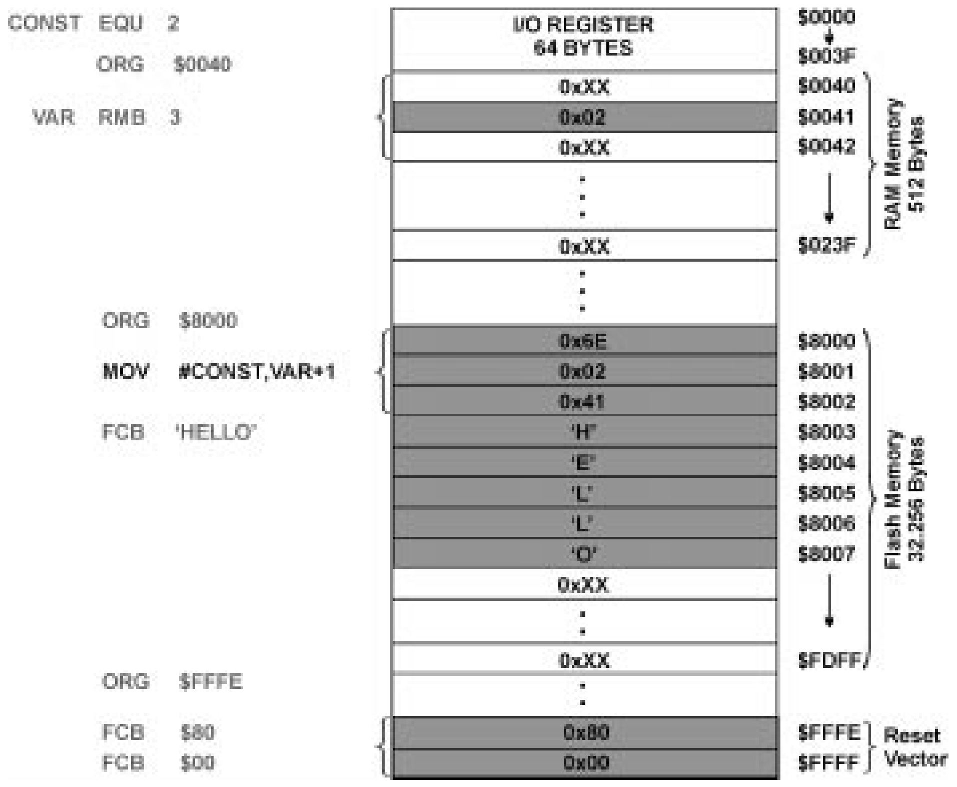
```
//data input from keyboard device
//data output to screen device

#include <stdio.h>

main()
 {
  char x;

  scanf  ("%c",&x);
  printf ("%c",x);
 }
```

Fig. 4. High-level programming example for data input/output from/to the outside world

registers, reset vectors, RAM, etc., particularly missing in higher-level code development. The devices used to receive data from the outside world should be assigned as input devices, and those used to send data to the outside world should be assigned as output devices. The MOV instructions located at \$8000 and \$8003 initialize the microcontroller's devices PORTB and PORTA as input and output ports to the outside world. Then the microcontroller executes the MOV instruction located at \$8006 to read the data input through the memory location \$0001 and to deliver the data output to the outside world through the memory location \$0000. Similar actions in high-level programming, like the example in Figure 4, are not involved with memory location accesses in the source code. The standard header file *stdio.h*, which is included during the compilation, contains the macros, structures, templates and other programming elements of the functions scan*f()* and print*f()* for accessing specific memory locations for data input from and output to the outside world, through the corresponding keyboard and screen device.

The representational example of Figure 5 describes how to use assembler directives to reserve and assign bytes in the microcontroller's data and program memory. The RMB pseudo-instruction is used to reserve a 3-byte variable (VAR), located at the very beginning of the data memory (according to the ORG \$0040 pseudo-instruction). The MOV instruction located at \$8000, assigns the second byte (\$0041) of the variable to the value of the constant (labelled as CONST). The pseudo-instruction FCB is used to assign the string 'HELLO' to the program memory locations \$8003-\$8007.

The previous examples are depicted in a reduced memory map scheme, as the extended memory map provided by the manufacturer does not provide an adequate picture for educational purposes. The pictures contain only basic information needed to explain low-level programming issues and the memory's sectors in the microcontroller. At this point it is worth noting that, on first attempts to build microcontroller-based tutoring around the computer architecture course, the lesson for the simulator's familiarization was placed in the second two-hour laboratory period. The students were asked to simulate simple assembly language programs, like previous ones, to realize concepts of low-level programming and to



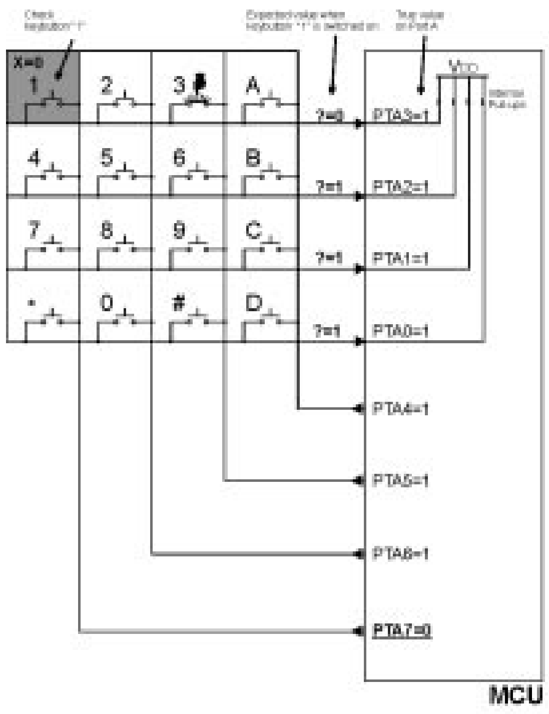
```
CONST EQU  2
      ORG  $0040
VAR   RMB  3

      ORG  $8000
      MOV  #CONST,VAR+1
      FCB  'HELLO'

      ORG  $FFFE
      FCB  $80
      FCB  $00
```

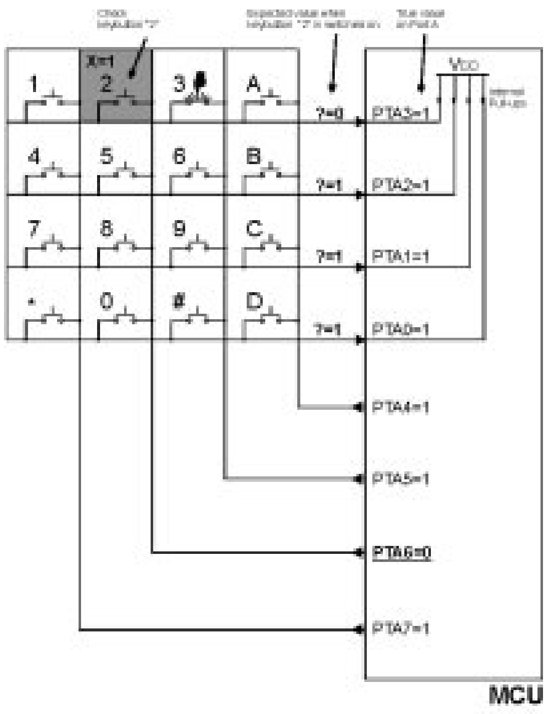
Fig. 5. Representational pictures: reserve/assign registers in data and program memory

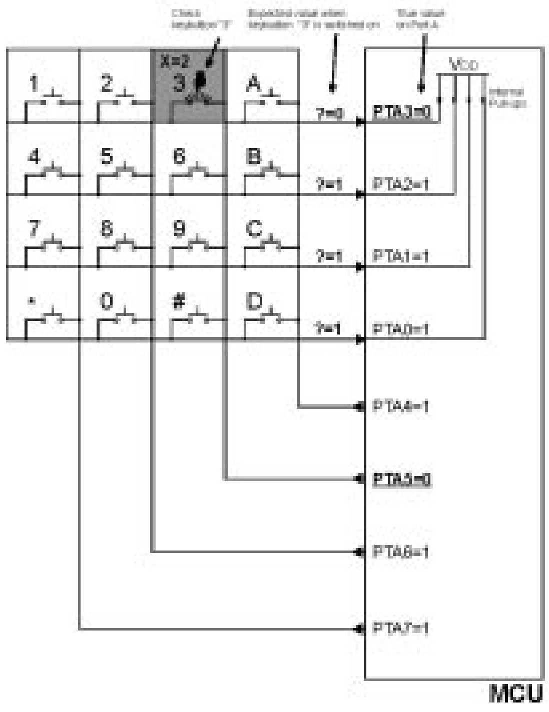
familiarize themselves with the simulation environment. The result was disappointing as the students became confused by the influx of information related both to low-level programming issues and simulation techniques. Spending a two-hour lesson (lab 2) on simple assembly language programs through representational pictures examples can be an efficient alternative in such situations. In addition, students can verify assembly language examples in the subsequent lesson (lab 3), where new learning issues only relate to simulation techniques.

Second part of the course: interpretational pictures

Figure 6 describes the steps of a matrix keyboard scanning procedure, until the switched on keybutton '3' is pressed. Three points are emphasized in the picture. The first one refers to

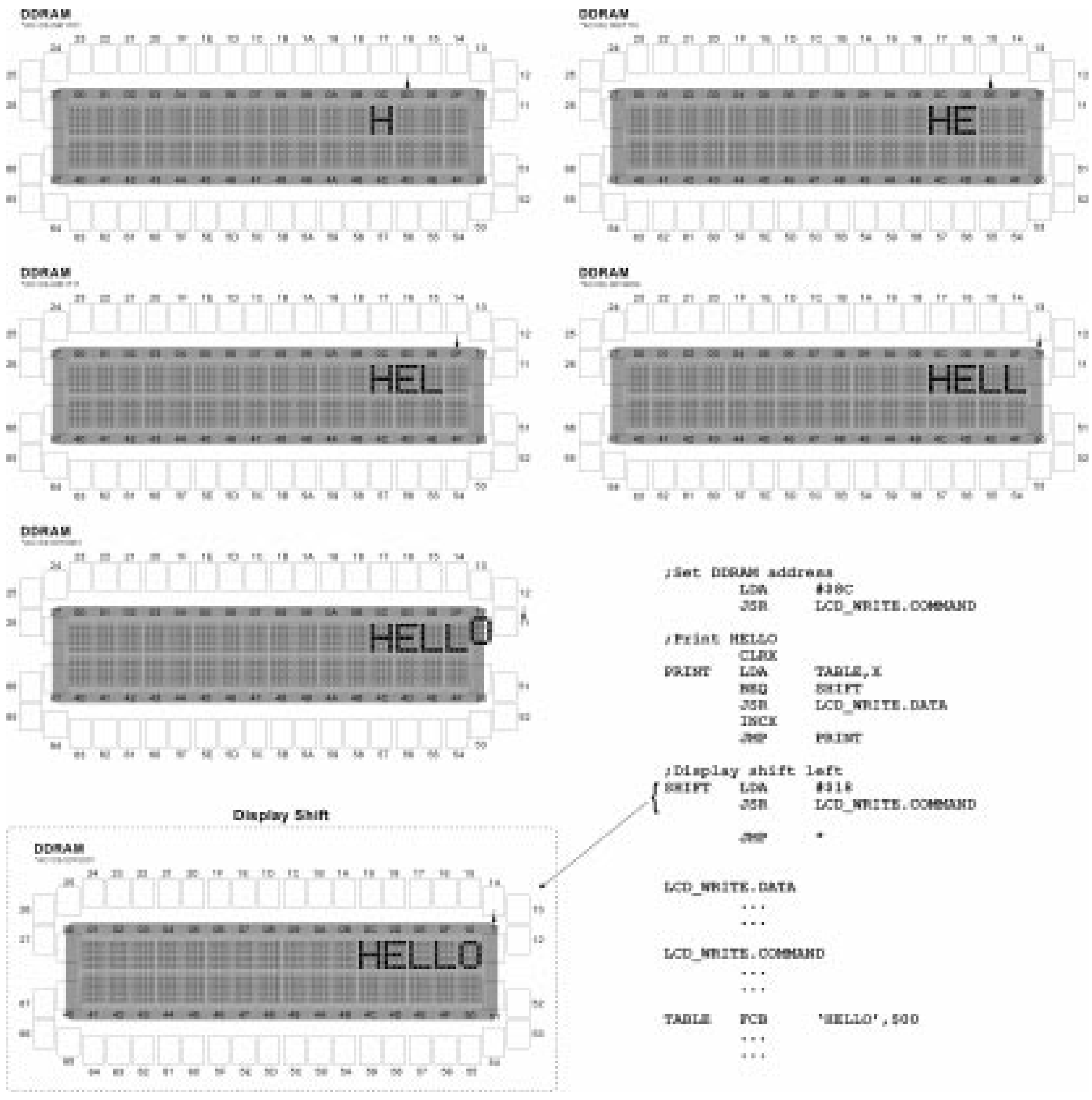






```
1    START   CLRX
2    SCAN    LDA      TABLE,X
3            STA      PORTA.DATA
4            CMPA     PORTA.DATA
5            BEQ      KEY.FOUND
6            INCX
7            CMPX     #!15
8            BLS      SCAN
9            JMP      START

10    KEY.FOUND
            ...
            ...

11    TABLE    FCB     %01110111
12             FCB     %10110111
13             FCB     %11010111
14             FCB     %11100111
15             FCB     %01111011
16             FCB     %10111011
17             FCB     %11011011
               ...
               ...
```

Fig. 6. Interpretational pictures: matrix keyboard scanning algorithm.



```
;Set DDRAM address
        LDA     #$8C
        JSR     LCD_WRITE.COMMAND

;Print HELLO
        CLRX
PRINT   LDA     TABLE,X
        BEQ     SHIFT
        JSR     LCD_WRITE.DATA
        INCX
        JMP     PRINT

;Display shift left
SHIFT   LDA     #$18
        JSR     LCD_WRITE.COMMAND

        JMP     *


LCD_WRITE.DATA
        ...
        ...

LCD_WRITE.COMMAND
        ...
        ...

TABLE   FCB     'HELLO',$00
        ...
        ...
```

Fig. 7. Interpretational pictures: LCD message printing and shifting

the table that carries the combinations of the expected switched on keybuttons, marked as the expected value (?=x).

The other two refer to the scanning algorithm, showing the relationship between the expected and true value on port A as well as the value of the index register (X), pointing to the keybutton that is scouted. Following the steps in the figure, the code analysis becomes easy to follow:

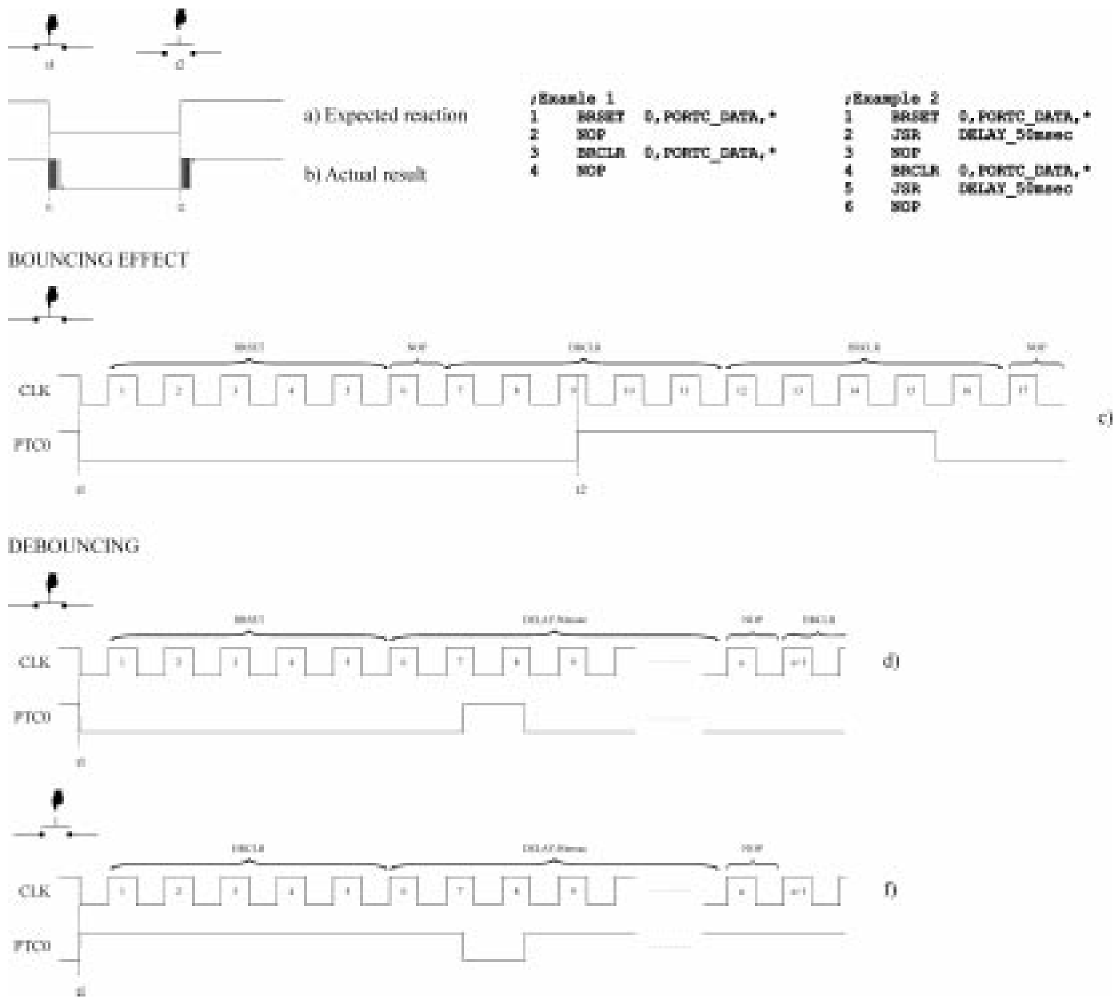
- *Step 1*: line 1 replaces the contents of X (index register low) with zeros. Line 2 assigns the accumulator with the first element of the array (0b01110111). Line 3 stores the contents of the accumulator in PORTA. The less significant lines of PORTA (PTA[3:0]) are assigned as inputs, while the most significant lines are assigned as outputs. The input lines in PORTA are changed according to the keybutton that is switched on. Since the keybutton '3' is switched on, the first element of the array, representing the expected value on PORTA when the keybutton '1' is switched on, does not match the true value on PORTA. Line 4 confirms this inequality (accumulator≠porta.data) and the microcontroller skips the execution of line 5.
- *Step 2:* the X register is incremented by one, pointing to the second element of the array (0b10110111). Lines 2–5 examine and confirm the inequality between the expected value and the true value on PORTA, when the keybutton '2' is assumed and keybutton '3' is switched on. Line 5 is skipped.
- *Step 3:* the X register is unary incremented, pointing to the third element of the array (0b11010111). Lines 2–5 examine and confirm the quality between the expected value and the true value on PORTA, when the keybutton '3' is assumed and switched on. Line 5 is executed.

The above example is difficult to understand even with use of a simulator. During the simulation, students alternate the lines in PORTA between the expected and the true value, actions that cause confusion. In addition, pictures that present only the interconnection between the microcontroller and the matrix keyboard, are appropriate only for students with an electrical/electronics bent. A parts-and-steps example makes a clearer link between the firmware and the hardware and surmounts the barriers of understanding.

A similar example in Figure 7 depicts the procedure for printing the message 'HELLO' on a liquid crystal display (LCD). In this example, the message is purposely printed out of the LCD's range, to highlight the relationship between LCD matrices and DDRAM locations. The latest step of the procedure indicates the way of revealing the whole message on the display by the execution of the appropriate command that shifts DDRAM one position to the left. In addition, the values of the address counter (AC) pointing to the next memory location, appear on every single step of the procedure.

Figure 8 presents a parts-and-steps example of the bouncing effect of mechanical switches in the code execution. The schematic a) presents the expected reaction of a pushbutton when the button is switched on (time t1) and switched off (time t2), while the schematic b) presents the actual result of the switch. The schematic c) presents the bouncing effect in the *'Example 1'* assembly code when the button is switched on. The schematics d) and f) present the debouncing procedure accomplished with a 50 msec delay, in the *'Example 2'* assembly code.

- *Example 1*: in Figure 8 c), the button is switched on at time t1 and the PTC0 signal goes low. The microcontroller executes the BRSET instruction in line 1 (first pulse of the clock), which finds the case *false*. After five clock cycles, the microcontroller fetches the subsequent NOP instruction in line 2, which is executed in one clock cycle. On the seventh pulse of the clock, the microcontroller executes the BRCLR instruction in line 3 and finds the case *true*. While the user still holds the button switched on, the mechanical switch bounces on time t2, causing the microcontroller to mistakenly scout the button switched off (12th pulse of the clock). As a result, the microcontroller finds the case in line 3 (BRCLR) *false* and executes the subsequent NOP instruction in line 4. Normally this program should execute line 2 of the code when the button switches on, and line 4 when the button switches off.



```
;Examle 1
1      BRSET   0,PORTC_DATA,*
2      NOP
3      BRCLR   0,PORTC_DATA,*
4      NOP
```

```
;Example 2
1      BRSET   0,PORTC_DATA,*
2      JSR     DELAY_50msec
3      NOP
4      BRCLR   0,PORTC_DATA,*
5      JSR     DELAY_50msec
6      NOP
```

Fig. 8. Interpretational pictures: bouncing effect on the execution of the assembly code

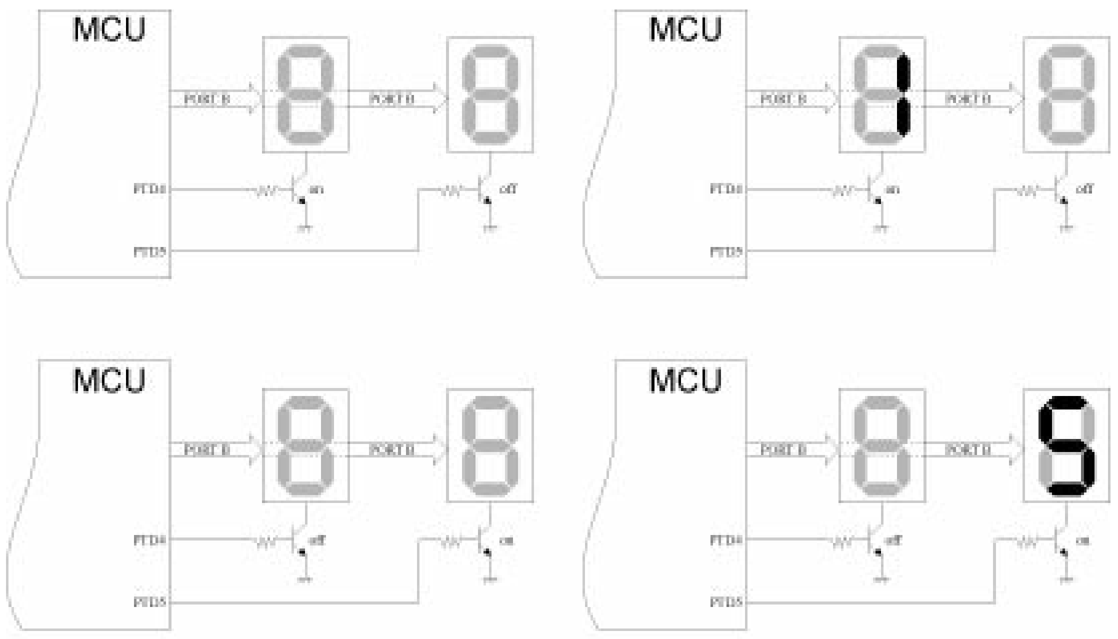
- *Example 2:* in Figure 8 d) the button is switched on at time t1 and the signal on the PTC0 signal goes low. The microcontroller executes the BRSET instruction in line 1, which finds the case *false*. After five clock cycles, the microcontroller delays 50 msec (line 2) until the bouncing phenomenon is extinguished. Then the microcontroller executes the NOP instruction in line 3 (nth pulse of the clock), and after that fetches the instruction BRCLR on the clock pulse n+1. The same procedure is described in Figure 8 f), when the button is released. Due to the delay of 50 msec, the bouncing phenomenon does not affect the flow of the execution of the code.

Figure 9 depicts an example of a multiplexing technique used to manage two seven-segment displays through a single output port (PORTB). The displays' common cathodes are activated/deactivated through the lines PTD4 and PTD5. In the first step, PORTB is cleared and PTD4 activates the first display (lines 1, 2 in the assembly code). In the second step the number '1' appears on the first display and the code delays (lines 3, 4). In the third step, PORTB is cleared once more, and the second display is activated through the PTD5 signal (lines 5, 6). In the fourth step, the number '5' appears on the second display and the code delays (lines 7, 8). Line 9 repeats the code execution for ever. Selecting an appropriate delay in the assembly code causes a non-discernible alternation of the displays, and the user sees the number '15' on the displays.

ASSESSMENT

Forty-seven students expressed an opinion of the computer architecture laboratory course. Table 2 summarizes the results of an 11-question assessment survey. The questionnaire was given to each student at the latest laboratory course, one week before the exam period.

The first four questions were selected to inquire into the students' attitude towards involvement in the topics of the course, i.e. knowledge of the embedded computer system architecture and operation (Question 1), the low-level programming concepts of microcomputer technology (Question 2), the overview of areas that microcomputers/microcontrollers are dealing with (Question 3), and the digital design aspects of implementation of a microcontroller's hardware (Question 4). Question 5 inquired whether involvement in embedded computers provided students with the opportunity to integrate topics that were covered in previously taken courses. Questions 6 and 7 explored the challenges implicit in the present approach, when the students are dealing with real



```
1  MAIN   CLR    PORTB.DATA
2         MOV    #%00010000,PORTD.DATA
3         MOV    #$30,PORTB.DATA
4         JSR    DELAY
5         CLR    PORTB.DATA
6         MOV    #%00100000,PORTD.DATA
7         MOV    #$5B,PORTB.DATA
8         JSR    DELAY
9         JMP    MAIN
```

Fig. 9. Interpretational pictures: driving two 7-segment displays from the same output device using multiplexing techniques

Table 2. Assessment survey

No	Questions	Very much (5)	Much (4)	Enough (3)	Shortly (2)	At all (1)	average
1	Did the computer architecture laboratory help you understand the internal structure and the principles of computers operation?	2	13	22	10	0	3,14
2	Did the computer architecture laboratory help you understand microprogramming and low-level programming issues?	3	21	13	10	0	3,36
3	Did the computer architecture laboratory provide you a significant experience in microcomputers and microcontrollers?	10	14	16	7	0	3,57
4	Did the computer architecture laboratory improve your skills on digital design?	3	10	16	11	7	2,8
5	Did the computer architecture laboratory help you understand topics that you learned in other courses?	6	19	18	3	1	3,55
6	Do you believe that working with hardware during the lessons increases the interest for the course?	14	15	12	4	2	3,74
7	Do you believe that one working with real hardware gains sense of accomplishment, instead of using only simulation tools?	30	14	1	2	0	4,53
8	Did the laboratory course motivate you to involve with similar issues (mC, mP, embedded systems, etc) in the future?	3	7	18	14	5	2,76
9	Did the laboratory course provide you the ability to involve with microcontroller applications in the future?	2	9	12	14	10	2,55
10	Do find satisfying the educational level of the laboratory course concerning other laboratory courses of the curriculum?	17	19	9	2	0	4,08
11	Did the computer architecture laboratory inspire your interest for the course concerning other laboratory courses you have attended?	12	17	14	2	2	3,74

hardware. Questions 8 and 9 explored student motivation and ability, particularly in regard to future involvement with embedded computer systems technology. Finally, Questions 10 and 11 examined whether students believed that the course had positive results on their education and interest in technological courses.

The average score on the first three questions appear in ascending order, which was expected given the educational approach. The students worked more on creating assembly language code for a microcontroller. In this case, the training benefit is dedicated more to microprogramming and microcomputers. The lack of hardware implementation during the lessons is reflected by the low score in Question 4. However, it was a conscious decision to leave hardware implementation out of lessons, as it would negatively affect students' progress on issues summarized in the first three questions. The slight difference in average score on Questions 1–3, indicates that the time-distribution on various topics of the course was properly defined, leaving no critical aspects out of the question. The average score on these questions might not be impressive, but it is promising given the students' academic backgrounds. It is worth remembering at this point that the students' background on related topics was rather limited, since the only prerequisite was that they should have previously attended the two-hour weekly theory and laboratory practice in digital design.

On Question 11, the students found the laboratory course inspiring, but with Question 8 they do not appear sufficiently motivated to want to become involved with microcontrollers, microprocessors, embedded systems, etc, in the future. However, this is a reasonable decision, given that most of the students would probably like to see themselves as software engineers. In Question 10, the students found the laboratory education level satisfying, although they don't believe that they acquired the ability to use the microcontroller in future applications (Question 9). This opinion too is fair enough, since this laboratory provides only an introduction to microcontrollers, so the course could not cover in-depth issues involved with real applications. Question 9 would probably earn more points in a subsequent senior level tutoring, where the students could work on real projects.

Positive results emerged from Questions 6 and 7, in which students emphasize that working with hardware increased their interest in the course and gave them a real sense of accomplishment. What is even more promising is the positive result in Question 5, where the students recognize that the course helped them to understand topics they had been taught previously. Indeed, introducing the hardware part of microcomputer and microprogramming issues, makes it easier to appreciate in depth the details of higher-level code development for a computer, and in some cases even arrive at a new personalized coding style.

CONCLUSIONS

This paper has presented a computer architecture laboratory course based on microcontroller tutoring that is addressed to students who have not specialized in electronic engineering. The selected approach was based on the design of appropriate educational equipment and an educational method that used representational and interpretational pictures. It emerged that representational and interpretational picture approaches are effective in surmounting several difficulties in student comprehension. Moreover, the students did not encounter serious difficulties in using the educational equipment, which also increased their interest in the course.

Acknowledgement —This work was partially supported by the research committee of Epirus Institute of Technology.

REFERENCES

1. B. Murovec and S. Kocijancic, A USB-based data acquisition system designed for educational purposes, *Int. J. Eng. Ed.* **20**, 2004, pp. 24–30.
2. F. J. F. Martín, J. C. C. Rodríguez, J. C. Á. Antón, J. C. V. Pérez, et al., An electronic instrumentation design project for computer engineering students, *IEEE Trans. Educ.* **48**, 2005. pp. 472–481.
3. A. del Río, J. J. Rodríguez-Andina, and A. A. Nogueiras-Meléndez, Learning microcontrollers with a CAI-oriented multi-micro simulation environment, *IEEE Trans. Educ.* **44**, 2001, pp. 197–211.
4. A. del Río and J. J. Rodríguez–Andina, UV151: a simulation tool for the teaching/learning the 8051 microcontroller, in *Proc. Frontiers in Education Conf.*, Kansas City, MO, pp. F4E/11-F4E/16. (2000).
5. H. B. Diab and I. Demashkieh, A computer-aided teaching package for microprocessor systems education, *IEEE Trans. Educ.* **34**, 1991, pp. 179–183.
6. H. Grunbacher, Teaching computer architecture/organization using simulators, in *Proc. Frontiers in Education Conf.*, Tempe, AZ, pp. 1107–1112. (1998).
7. N. L. V. Calazans and F. G. Moraes, Integrating the teaching of computer organization and architecture with digital hardware design early in undergraduate courses, *IEEE Trans. Educ.* **44**, 2001, pp. 109–119.
8. D. C. Hyde, Teaching design in a computer architecture course, *IEEE Micro* **20**, 2000, pp. 23–28.
9. G. M. Brown and N. Vrana, A computer architecture laboratory course using programmable logic, *IEEE Trans. Educ.* **38**, 1995, pp. 118–125.
10. Wahyudi, M. J. E. Salami and A. Albagul, Development of a microcontroller-based control system with a hardware-in-the-loop (HIL) method for control education using matlab/simulink/xPC target, *Int. J. Eng. Ed.* **21**, 2005, pp. 846–854.
11. J. O. Hamblen, A. Parker and G. A. Rohling, An instructional laboratory to support microprogramming, *IEEE Trans. Educ.* **33**, 1990, pp. 333–336.
12. J. W. Jeon, A microprocessor course: designing and implementing personal microcomputers, *IEEE Trans. Educ.* **43**, 2000, pp. 426–433.
13. A. Kutlu, Microlab: a web-based multi-user remote microcontroller laboratory for engineering education, *Int. J. Eng. Ed.* **20**, 2004, pp. 879–885.
14. D. F. Hanson, A microprocessor laboratory for electrical engineering seniors, *IEEE Trans. Educ.* **E-24**, 1981, pp. 8–14.
15. T. K. Hamrita and R. W. McClendon, A new approach for teaching microcontrollers courses, *Int. J. Eng. Ed.* **13**, 1997, pp. 269–274.
16. (ABET) Accreditation Board of Engineering and Technology. Online. Available: http://www.abet.org/
17. J. R. Levin, On functions of pictures in prose, in F. J. Pirozzolo & M. C. Wittrock (eds.), *Neuropsychological and Cognitive Processes in Reading*, pp. 203–228 Academic Press, New York (1981).
18. D. Kirsh, Why illustrations aid understanding, in *Proc Int. Workshop on Dynamic Visualizations and Learning*, Tubingen, Germany, (2002).
19. R. N. Carney and J. R. Levin, Pictorial illustrations still improve students' learning from text, *Educ. Phychol. Review* 14, pp. 5–26 (Mar. 2002).
20. R. E. Mayer and J. K. Gallini, When is an illustration worth ten thousand words?, *J. Educ. Psychol.* **82,** 1990, pp. 715–726.

Dimosthenis E. Bolanakis was born in Crete, Greece, in November 1978. He received a B.Sc. degree in electronic engineering from the Department of Electronics, Thessalonikis Educational Institute of Technology (TEI), Thessaloniki, Greece, in 2001 and his M.Sc. degree in modern electronic technologies from the Department of Physics, University of Ioannina (UoI), Ioannina, Greece in 2004. He is currently pursuing a Ph.D. degree in Physics Department, UoI, Ioannina, Greece. During his M.Sc. education, he participated as a teaching assistant in the elective graduate course Microcontrollers—Microprocessors at the Physics Department, UoI, for the academic years 2003–2004. From 2003 till today, he has taught Computer Architecture lab training at the Department of Communications, Informatics and Management, Epirus Educational Institute of Technology (TEI.), Arta, Greece. At present his main interest is data acquisition systems.

Euripidis Glavas received a B.Sc. degree in the Physics Department of the University of Ioannina, Ioannina, Greece, in 1983, and the Ph.D. degree from Sussex University, UK, in 1989. He has worked as a Research Associate at the University of Sussex, the University of Liverpool and the Democritus University of Thrace. In 2001, he joined the Department of Communications, Informatics and Management of the Epirus Educational Institute of Technology (TEI.), Arta, Greece, where he is an associate professor. Currently, he is teaching an undergraduate course in Computer Architecture at the Department of Communications, Informatics and Management as well as a postgraduate course in microprocessors architecture and assembly language at the Physics Department, University of Ioannina, Ioannina, Greece. His primary research interests include Computer Architecture, Computers in Education, Microprocessors and Microcomputers.

Giorgios A. Evangelakis received his B.Sc. degree in the Physics Department, Aristotelian University of Thessaloniki, Thessaloniki, Greece in 1980 and his Ph.D. degree from the University of Nancy I, Nancy, France in 1989. He is an associate professor at the Physics department, University of Ioannina, Ioannina, Greece, and currently teaches the elective graduate course Microcontrollers—Microprocessors. His primary research interests are computer simulation techniques and applications.