ELECTRONICS AND ELECTRICAL ENGINEERING

ISSN 1392 – 1215 2010. No. 2(98)

ELEKTRONIKA IR ELEKTROTECHNIKA

SYSTEM ENGINEERING, COMPUTER TECHNOLOGY

T 120

SISTEMŲ INŽINERIJA, KOMPIUTERINĖS TECHNOLOGIJOS

A Highly Interactive PC based Simulator Tool for Teaching Microprocessor Architecture and Assembly Language Programming

N. Topaloglu

Department of Electronics and Computer Education, Technical Education Faculty, Gazi University, Teknikokullar, 06500, Ankara, Turkey, tel.: +90 312 202 8575, e-mail: Nurettin@gazi.edu.tr

O. Gürdal

Department of Electrical Education, Technical Education Faculty, Gazi University, Teknikokullar, 06500, Ankara, Turkey, tel.: +90 312 202 8548, e-mail: ogurdal@gazi.edu.tr

Introduction

The principal author of this paper has taught advanced undergraduate level courses in computer sciences for the past six years. Lab-Volt electromechanical μP experimental sets [1] have been used for microprocessor (μP) and computer architecture course at the University of Gazi in Ankara, Turkey. As the number of the students in classrooms is crowded and the number of the sets is insufficient, the students are grouped into two teams. Attempting to increase the number of the sets faced cost considerations [2, 3]. The student having designed the software component of a project may not be able to access to the laboratory because of schedule/security difficulties [3]. Like most computer system courses, Assembly Language Programming (ALP) is being used as a vehicle to understand the interrelationship and interactions between the different components of a computer system [4]. The programs in ALP written by the user are translated to machine language by using a hand assembler or third party assembler and are entered to experimental sets [5, 6]. During the execution of the programs, the students are unable to see what steps are taken in the hardware units and to comprehend the consequences [7-10]. This is especially the case when teaching the matters regarding to I/O operations. The students are also unable to get sufficient help materials and supervision during the laboratory sessions [2, 3, 11]. As a result, the success of the students which has been monitored stayed below a certain level. In proposing new methods when teaching/learning μP architecture and ALP, the use of computers and functional simulators as an assisting tool is emphasized [4, 12]. The use of a simulator has two advantages: the first one is to develop a suitable simulator for desired μP except commercial and standard types, the second one is to get good results by using simulator for visualization and debugging. In addition, the programs written for a processor are able to be executed in computers having different types of processors [4]. This enables the students to use the simulators at any environment.

The μP simulators used presently either simulate a μP other than the one available in the labs or are unable to simulate all the functions of a μP [6], especially in case of controlling peripheral devices which are connected to the system. A PC and windows based simulator tool is developed in parallel with the Lab-Volt electromechanical μP experimental set with 6502 [13]. When developing this simulator, curricula's of computer science department at the Technical Education Faculty, the University of Gazi in Ankara, Turkey and the other appropriate departments have been taken into consideration.

The simulator which has been used for two semesters has attracted great attention by the students (and educators as well). They preferred this software to the electromechanical experimental sets. As a result, success of the students monitored in μP course increased compared to the previous years.

The μP simulator named as the **VISUAL 6502** is designed as having single main menu from which all the functions are reachable. The C++ Builder Compiler has been used in order to develop the simulator. It occupies nearly 3.230 MB in hard disc and 10,764 KB in RAM. A copy of the simulator software executable in all PCs can be downloaded from the address given in the reference [14].

The System Components

By the **VISUAL 6502** μP simulator tool, most of the problems mentioned in the introduction section have been eliminated. This virtual software with developed features is useful tool for the related academical and industrial departments.

The simulator consists of:

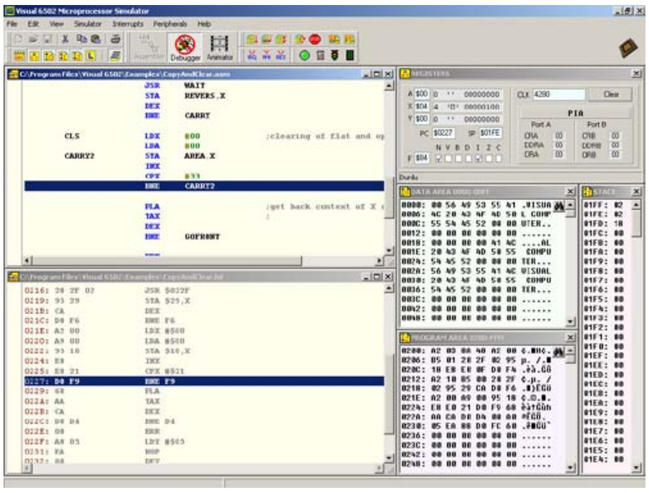
a) Local editor: It is a simple word processor. No other tool is needed to write assembly programs. They can be saved with 'ASM' extension after being edited by the user.

b) Assembler: The assembler has the ability to process the whole instruction set of the 6502 (i.e. 56 instructions in total). The µP instruction set is translated into the corresponding machine language. Upon the execution of the program, a '*.HEX' file containing executable program codes, a '*.LST' disassembler file having the addresses corresponding to flags and codes used in the program, and an empty '*.LOG' file is created to be used during the debugging process.

c) Debugger: The machine coded instructions within the '*.HEX' file can be executed by the debugger step by step, at user selected speeds. The effect of the instructions on the registers and memory areas can be monitored in a window on the simulator so that the students are able to see the effects of instructions on the registers, memory and buses.

d) Animator: All the units of the processor normally visible or invisible to the programmer with memory and I/O units are animated in this section. How the command codes and the data are processed is visually animated in 'fetch' and 'execute' steps and the students are provided to understand the operation steps of the processor.

e) Virtual I/O Elements: This is where our simulation platform differs from the rest. To teach µPs in controlling mechanisms, initially some I/O elements are embedded into the simulator which we plan to extend as we evolve. An 8 bits switch set as input element, 8 bits LED set as output element, two traffic lights and a 6 digits 7-segments display, a simple scope, and a step motor are simulated at this stage.



```
                JSR     WAIT
                STA     REVERS,X
                DEX
                BNE     CARRY

CLS             LDX     #00             ;clearing of flat and up
                LDA     #00
CARRY2          STA     AREA,X
                INX
                CPX     #33
                BNE     CARRY2

                PLA                     ;get back context of X
                TAX                     ;
                DEX
                BNE     GOFRONT
```

```
0216: 20 2F 02      JSR $022F
0219: 95 29         STA $29,X
021B: CA            DEX
021C: D0 F6         BNE F6
021E: A2 00         LDX #$00
0220: A9 00         LDA #$00
0222: 95 10         STA $10,X
0224: E8            INX
0225: E0 21         CPX #$21
0227: D0 F9         BNE F9
0229: 68            PLA
022A: AA            TAX
022B: CA            DEX
022C: D0 D4         BNE D4
022E: 00            BRK
022F: A0 05         LDY #$05
0231: EA            NOP
0232: 88            DEY
```

```
A $00  0  ''   00000000      CLK 4280
X $04  4  ''   00000100
Y $00  0  ''   00000000          Port A        Port B
   PC $0227   SP $01FE       CRA   00      CRB   00
       N V B D I Z C         DDRA  00      DDRB  00
F $84                        ORA   00      ORB   00
```

```
0000: 00 56 49 53 55 41  .VISUA
0006: 4C 20 43 4F 4D 50  L COMP
000C: 55 54 45 52 00 00  UTER..
0012: 00 00 00 00 00 00  ......
0018: 00 00 00 00 41 4C  ....AL
001E: 20 43 4F 4D 50 55   COMPU
0024: 54 45 52 00 00 00  TER...
002A: 56 49 53 55 41 4C  VISUAL
0030: 20 43 4F 4D 50 55   COMPU
0036: 54 45 52 00 00 00  TER...
003C: 00 00 00 00 00 00  ......
0042: 00 00 00 00 00 00  ......
0048: 00 00 00 00 00 00  ......
```

```
0200: A2 03 8A 48 A2 00  ¢.■H¢.
0206: 85 01 20 2F 02 95  µ. /.■
020C: 18 E8 E0 0F D0 F4  .èà.Ðô
0212: A2 10 85 00 20 2F  ¢.µ. /
0218: 02 95 29 CA D0 F6  .■)ÊÐö
021E: A2 00 A9 00 95 10  ¢.©.■.
0224: E8 E0 21 D0 F9 68  èà!Ðùh
022A: AA CA D0 D4 00 A0  ªÊÐÔ. 
0230: 05 EA 88 D0 FC 60  .ê■Ðü`
0236: 00 00 00 00 00 00  ......
023C: 00 00 00 00 00 00  ......
0242: 00 00 00 00 00 00  ......
0248: 00 00 00 00 00 00  ......
```

```
01FF: 02
01FE: 02
01FD: 18
01FC: 00
01FB: 00
01FA: 00
01F9: 00
01F8: 00
01F7: 00
01F6: 00
01F5: 00
01F4: 00
01F3: 00
01F2: 00
01F1: 00
01F0: 00
01EF: 00
01EE: 00
01ED: 00
01EC: 00
01EB: 00
01EA: 00
01E9: 00
01E8: 00
01E7: 00
01E6: 00
01E5: 00
01E4: 00
```

Fig. 1. General view of the VISUAL 6502 µP simulator

In the simulator, while 56 commands and 13 different addressing modes are being used, enhanced assembler instructions are included as well. Errors encountered after the translation of the program into machine codes by the assembler are dealt with at this stage. Program errors can be eliminated by two approaches: the first method is elimination of syntax errors caused by misused instructions. They may occur while assembling the program and can be reached by clicking on the error code. In the second method which corresponds to the logical errors that may occur while debugging -the program- the elimination of the errors depend on the experience of the programmer.

The simulator with the simulated memory and I/O systems forms the kernel of a simple computer. The graphical output of the **VISUAL 6502** µP simulator can be seen in Fig.1.

Many functions of the simulator are reachable from this window which is briefly described below;

1) Main Control Menu: This menu has six groups consisting of

- buttons being used for editing processes,
- 256 bytes memory of which 64 KB is zero page (data area), 256 bytes is first page (stack area) and the rest is program page reserved for buttons dumping the screen and a button is for closing memory windows and log windows,
- main function starting buttons for the assembler, the debugger and the animator,
- buttons for selecting the speed of the program, and stopping and restarting program with memory erasing buttons,
- buttons with which interrupt (IRQ, NMI and RES) types are selectable,
- buttons with which virtual I/O elements are selectable.

2) Editor Area Window: This area is used for writing the programs. Addressing the labels which is impossible in the electromechanical experimental set is implemented in this area. The command being processed during execution of the program is highlighted.

3) Machine Codes Window: After the program has been written in the editor window, it is debugged. Codes, addresses (effective address-offset) and labels of the program can be seen in this area. Programs purged free from instructions, mnemonics, operation codes and operand areas are displayed in this window. The horizontal stripe working in the editor window is also used here to highlight the position and both highlights in two windows work in parallel.

4) Register Window: 6502 processor has three general (A, X and Y) and three private purpose (PC, SP and flags) registers visible by the user. In addition to those, the invisible registers of A and B port registers of 6520 PIA I/O integrated circuit (CRA/CRB, DDRA/DDRB and ORA/ORB) are located in this window.

5) Memory Area: 64 KB memory area is divided into three sections as data area, stack area and program area. While the stack area shows 0100H and 01FFH address contents, the data area shows 0000H and 0100H address space and the program area shows 0200H and FFFFH address space, and I/O addresses also take place in the program area. Contents of the memory region selected in the data and the program area can be displayed. In the memory windows, hex contents and their ASCII equivalents can be monitored synchronously.

Details of the Data Format

When writing a program in ALP, a data description format has been developed. As the 6502 has an 8-bit processor, a bit oriented data description has been implemented taking this into consideration. As in developed assemblers the data to be saved in the data area in memory starts with a pointer instruction:

```
      .DATA  $20 ;starting address of the TABLE
TABLE .BYTE  25,$FA,30,$71,127,$2E
      .BYTE  63,15,22,$45,29,00
      .DATA  $30 ;starting address of the LIST
LIST  .BYTE  10,$3B,$54,33, "COMPUTER"
```

The data can be put in the memory area with the ‘*.DATA $address*’ definition in more than one region. Here ‘.DATA’ definition is an instruction describing the starting address of the data area. Data in specified areas can be described as hexadecimal, decimal or string while being represented in bit format.

The Representation of the Instructions

The instruction format is also “byte” oriented as in data. 13 different addressing modes can be used with instructions. Direct/indirect addressing, register/memory operand, absolute/relative addressing and indexing addressing modes can be selected. The addresses described as labels, which can be used as label defined in operand area and the labels can be given displacement values as well.

Instructions in code area can be described with addresses in enhanced assembler format:

```
   .CODE $0200         ;starting address of the codes
START  LDX #00
ADDT   ADC TABLE,X
       INX
       CPX #05
       BNE ADDT
       STA $40
       BRK
       END             ; end of the program
```

In the program, ‘*.CODE $address*’ shows the starting address for the program while the END directive points the end of the program. The usage of labels during the program development stage ensures the relocation of the addresses to be corrected after the modifications. This is impossible when using the electromechanical experimental sets.

Apart from the ordinary arithmetic, logic and jumping operations, four general shifting and returning operation instructions such as stack operations, sub-program calling and reversing instructions can be used with their full functions.

The Assembler

The students can easily be mistaken when calculating offsets in defining zero page and absolute addresses in the operand portion. The labels might be used instead of base addresses at absolute and index addressing modes. Then the labels might be converted to their relative addresses. Relative addressing is an efficient way of readdressing and is obtained by using the PC as ‘base’ address. Memory address is computed by adding one of index register value (X or Y). When formulated, content of the register is represented by:

Effective address = base address + index register.

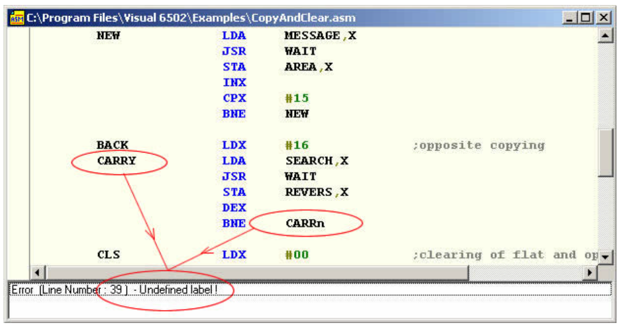
```
TABLE    .DATA $20
         .BYTE 05,10,15,20,25
         .CODE $0200
         .
         ADC TABLE,X
         .
         END
```

In the instruction sequence above, the address corresponding to the TABLE is 0020H. If X=03H then the operand (effective address) of the ADD command will be TABLE+X=0020H+03H=0023H. Here the address of the TABLE label, 0020H is added to X register 03H from which the effective address is obtained.

In the assembler, the ‘*.BYTE*’ definition is proposed to give the students the ability to define data. By ‘*.BYTE*’ decimals, hexadecimals and strings can be defined.

```
      .DATA $40
TABLE .BYTE 10,255,00,45 ;decimal number definitions
      .BYTE $10,$FF,$5C,$22;hex. number definitions
MSG   .BYTE "BY THE TIME"   ; string definitions
MIXED .BYTE 25,$25, "ELECTRONICS"  ;complex
data definition
```

The ‘*.DATA $address*’ definition is sufficient to define the data area. Except the data area, in the operand area or in the definitions of the ‘DATA’ or the ‘CODE’ segment, the numerical values with ‘$’ mark are defined as addresses. In the program, the ‘*.CODE $address*’ definition and the ‘END’ are the instructions showing the starting point of the memory area to which program codes are saved, and end of the program respectively.



C:\Program Files\Visual 6502\Examples\CopyAndClear.asm

```
    NEW       LDA    MESSAGE,X
              JSR    WAIT
              STA    AREA,X
              INX
              CPX    #15
              BNE    NEW

    BACK      LDX    #16              ;opposite copying
    CARRY     LDA    SEARCH,X
              JSR    WAIT
              STA    REVERS,X
              DEX
              BNE    CARRn

    CLS       LDX    #00              ;clearing of flat and op
```

```
Error  (Line Number : 39 ) - Undefined label !
```

Fig. 2. Editor and error messages window

By pressing the assembler button after the program being edited in the editor;

1. a ‘.HEX’ file having hexadecimal format command codes,

2. a '.LST' file including program, machine and its final target
3. a '.LOG' file to be filled when the program is run are automatically created.

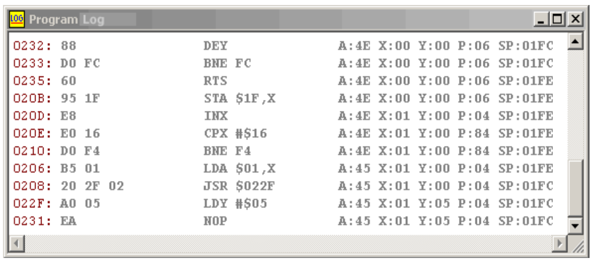
If there are syntax errors in the program before the creation of these files, a pop up window will appear under the editor window showing error codes and their possible meanings. When the user clicks the mouse on these error codes, the cursor directly goes to the corresponding line where the error is as seen in Fig. 2. This process is extremely useful for the students when writing programs.

The Debugger

One of the common complaints of the students is that the difficulty of understanding the way the data passes through registers, memory or I/O units when the program is running and monitor especially in logical processes how the inter unit operation is effected in what cycle, and how the synchronization is established in the case of I/O operation. The highly interactive animation tools play a vital role here: it animates all the events in a clear and understandable way, enabling the student to see what he/she actually meant by a specific I/O instruction. The operations could easily be understood as the debugger is able to show instantly which registers have and 64KB full memory area in real time. Register/memory relationship could easily be interpreted. The results of mathematical and logical processes could be understood by examining the bits of the flag register.

The errors in the program can also be monitored by pressing the debug button, at user selectable speeds. A sequence of specifications of the debugger as below enables the students to find errors and to write more effective and compact programs. The actions listed are such as:

- Adding breakpoints,
- Use of IRQ, NMI and RES interrupts,
- Step by step program execution,
- Modifying register/memory contents,
- Memory dumping



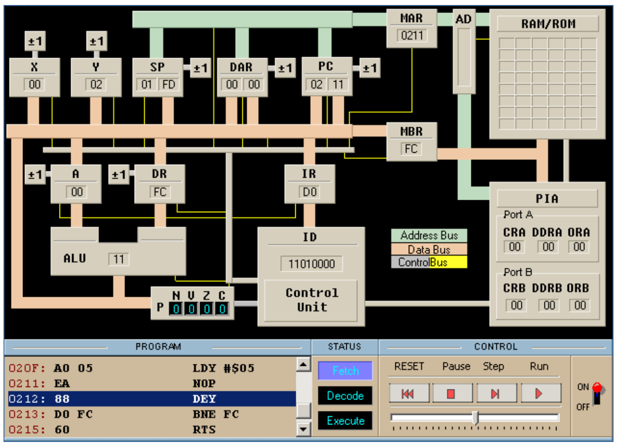
```
0232: 88          DEY          A:4E X:00 Y:00 P:06 SP:01FC
0233: D0 FC       BNE FC       A:4E X:00 Y:00 P:06 SP:01FC
0235: 60          RTS          A:4E X:00 Y:00 P:06 SP:01FE
020B: 95 1F       STA $1F,X    A:4E X:00 Y:00 P:06 SP:01FE
020D: E8          INX          A:4E X:01 Y:00 P:04 SP:01FE
020E: E0 16       CPX #$16     A:4E X:01 Y:00 P:84 SP:01FE
0210: D0 F4       BNE F4       A:4E X:01 Y:00 P:84 SP:01FE
0206: B5 01       LDA $01,X    A:45 X:01 Y:00 P:04 SP:01FE
0208: 20 2F 02    JSR $022F    A:45 X:01 Y:00 P:04 SP:01FC
022F: A0 05       LDY #$05     A:45 X:01 Y:05 P:04 SP:01FC
0231: EA          NOP          A:45 X:01 Y:05 P:04 SP:01FC
```

Fig. 3. Program diary window

Apart from the specifications listed above, the record of every instant of execution of the program is stored in a '.LOG' file which enables the students to reiterate the events developed in detail at home or school as seen in Fig. 3. In the debugger, during the execution of the program the instruction being processed is highlighted in the editor and the disassembler windows by a horizontal line. This enables the program and its logic to be understood clearly.

The Animator

The animator animates the signalization of command input/output between memory, I/O units and communication paths together with the basic units normally visible or invisible to the programmer. As seen in Fig. 4, the animator clarifies all the events by giving the ability to monitor what is going on within computer.

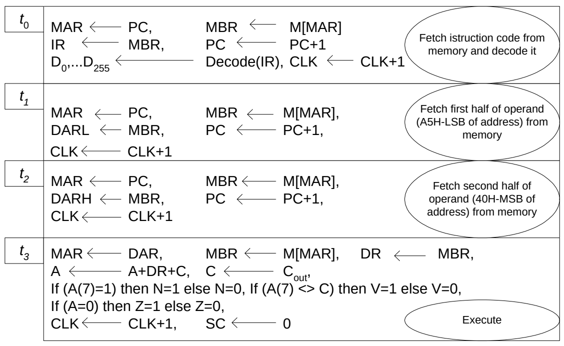


```
020F: A0 05       LDY #$05
0211: EA          NOP
0212: 88          DEY
0213: D0 FC       BNE FC
0215: 60          RTS
```

Fig. 4. The Animator window

The animator is highly user friendly with specifications listed below:

- Presence and function of invisible registers like MAR, MBR, DAR, DR and IR,
- Monitoring the command fetch/decode (t_0), fetch operand (t_1-t_2) and execute (t_3) in a clear manner, the command in this duration takes a time of 2 cycles min. and 7 cycles max,
- Interpretation of commands in RTL language. The registers used by a ADC (ADC $40A5) command addressed as absolute continuing four machine cycles are seen in Fig.5,
- The relationships between processor, memory, several I/O units and communication paths.


t_0
Fetch istruction code from memory and decode it
t_1
Fetch first half of operand (A5H-LSB of address) from memory
t_2
Fetch second half of operand (40H-MSB of address) from memory
t_3
Execute

t_0: $MAR \leftarrow PC$, $MBR \leftarrow M[MAR]$
$IR \leftarrow MBR$, $PC \leftarrow PC+1$
$D_0,\ldots D_{255} \leftarrow Decode(IR)$, $CLK \leftarrow CLK+1$

t_1: $MAR \leftarrow PC$, $MBR \leftarrow M[MAR]$,
$DARL \leftarrow MBR$, $PC \leftarrow PC+1$,
$CLK \leftarrow CLK+1$

t_2: $MAR \leftarrow PC$, $MBR \leftarrow M[MAR]$,
$DARH \leftarrow MBR$, $PC \leftarrow PC+1$,
$CLK \leftarrow CLK+1$

t_3: $MAR \leftarrow DAR$, $MBR \leftarrow M[MAR]$, $DR \leftarrow MBR$,
$A \leftarrow A+DR+C$, $C \leftarrow C_{out}$,
If (A(7)=1) then N=1 else N=0, If (A(7) <> C) then V=1 else V=0,
If (A=0) then Z=1 else Z=0,
$CLK \leftarrow CLK+1$, $SC \leftarrow 0$

Fig. 5. Execution of ADC $40A5 command line in RTL language

The Virtual I/O Elements

In learning assembly based programming, one of the important subject is to control external peripheral elements together with the ability to view interior events in µPs. Students usually experiment difficulties in such cases especially when executing the programs on hardware units. This is mainly due to not having the ability of visualizing the internals of the µP at the five of I/O events. Main task of a µP is to control an element according to time intervals specified. The students are required to know how a µP can be used in real life events such as controlling some peripheral elements. For this purpose, I/O elements like virtual LED, switch, traffic lights and a 7- segments display are simulated. Interactive peripheral displays corresponding to some of these peripherals are given in Fig. 6(a) and 6(b) respectively. For controlling these, 6520 PIA integrated circuit is selected as I/O unit.

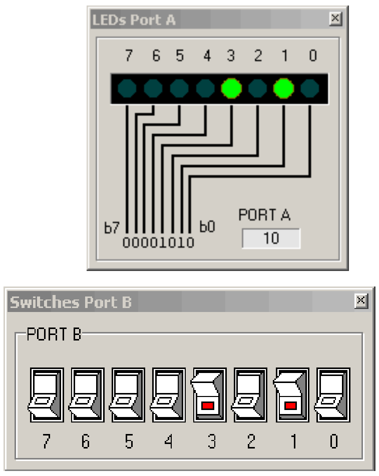


Fig. 6(a). LED and microswitches group

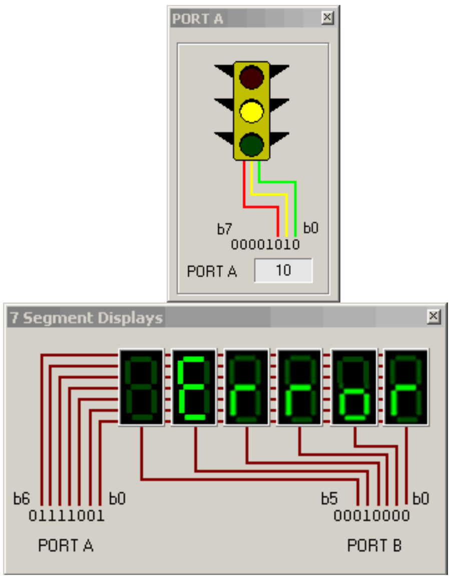


Fig. 6(b). Traffic lights and 6 digits 7-segments display

The logical errors which may not be visible in the programs can become apparent on visual elements. That's way the simulator has embedded I/O elements to give the students the ability to design real life controlling mechanisms, therefore to understand the way of that a µP communicates with the peripheral devices.

Two groups of microswitches are selected as input elements, and 2 groups of LEDs as output elements, traffic lights and 6 digits 7-segments display, a simple scope and a step motor are designed for 8 bits A and B ports of 6520 PIA I/O integrated circuit. The values on the displays can be monitored as either decimal or binary numbers.

Conclusions

Today, the developments in computer resources and multimedia tools are helpful in order to develop educational software for every steps of the education. Using these facilities, a highly interactive µP simulator with the special focus on the internal details of the I/O operations has been developed to provide the students comprehend topics thought in µP programming courses in a cheap and effective manner. The contribution of the study towards the education can be listed as:

- All the data flow in the machine is visual and visible.
- There is no need to enter the machine codes manually as the machine has a local assembler.
- Logical errors can be eliminated during the development of the program. This saves time as reentering the program is not needed.
- The applications support student focus and help, and the support may be given by help menu or online on the internet.
- The internal processing of the µP commands is designed to be monitored step by step by the user.
- There is no need intense manual works.
- Timing of microprogram is easy to be understood.
- The limited I/O operations of such systems are extended to help the students on this topic.

Its financial side of the contribution can be listed as:

- While the simulators can be used on any PC, electromechanical experimental sets are only available in the labs.
- Its cost is very low.
- There are no breakdown problems and no need for maintenance and technician support.
- No or little extra costs for the students.
- The simulator is ready to use every time but the labs accommodating electromechanical experimental sets are not ready to be reached every time for security and scheduling considerations.

The tool developed shows to be used as an application tool in industrial schools and where the µPs are taught, since it meets all the requirements of a µP which are present in an experimental set for teaching computer architecture and animates all the events in a computer visually and graphically. As a result, teaching µP architecture and ALP is provided in quick and efficient way.

Future works may include transferring of machine codes in memory of simulator to electromechanical experimental set by an emulator and retransferring of these from the set to simulator by which a computer interfaced electromechanical experimental education set can be established. Some extra virtual application I/O elements for the user can be added to the system.

The interactive **VISUAL 6502** μP simulator which is interactive, flexible and user friendly has been tested for two semesters in the μP lab. at the University of Gazi. According to our observations, the students find the use of simulators easier than the electromechanical sets. In addition, the use of the software has increased the overall success of the students taking part of the course.

References

1. **Buck Engineering Co. Inc.** Microprocessor Concepts and Applications. – Lab-Volt, USA. − 1994.
2. **del Rio A., Andiana J. J. R.** UV151: A Simulation Tool For Teaching // Learning The 8051 Microcontroller, Frontiers in Education Conference, FIE 2000, 30th Annual. − 2000. − Vol. 2. − P. F4E/11-F4E/16.
3. **Smith M. R., Cheng M.** Use of Virtual (simulated) hardware devices in microprocessor laboratories and tutorials // Frontiers in Education Conference, FIE’96, 26th Annual Conference. − 1996. − Vol. 3. − P. 1181–1185.
4. **Pearson M., Armstrong D., McGregor T.,** Design of a Processor to Support the Teaching of Computer Systems // Proceedings of the First IEEE International Workshop on Electronic Design, Test and Applications (DELTA’02). − 2002. − P. 240−244.
5. **Lovergrove W. P.** A Microprocessor Trainer Simulator // Proc. of the 26th Frontiers in Education Annual Conference. − 1996. − Vol. 2. − P. 506−509.
6. **Caldwell C. W., Andrews D. L., Scott S. S.** A Graphical Microcomputer Simulator for Classroom Use // Frontiers in Education Conference. – 1995. − Vol. 2. − P. 3b3.9–3b312.
7. **Robbins S., Robbins K. A.** A Microprogramming Animation // IEEE Transaction on Education. − 1998. − Vol. 41, No. 4. − P. 293−300.
8. **Wick C. E.** Teaching Embedded Computer Systems with a Windows-Based Simulator // Frontiers in Education Conference, FIE’96. – 1996. − Vol. 1. − P. 242–245.
9. **Beaumont M., Jackson D.** Visualisation as an Aid to Low-Level Programming // Frontiers in Education Conference, 27th Annual Conference. Teaching and Learning in an Era of Change. − 1997. − Vol. 3. − P. 1158−1163.
10. **Diab H. B., Demashkieh I.** A Computer-Aided Teaching Package for Microprocessor Systems Education // IEEE Transaction on Education. − 1991. − Vol. 34, No. 2. − P. 179−183.
11. **Handerson W. D.** Animated Models for Teaching Aspects of Computer Systems Organization // IEEE Transactions on Education. − 1994. − Vol. 37, No. 3. − P. 247−256.
12. **Martins Carlos A. P. S. et al**. A New Learning Method of Microprocessor Architecture // 32th ASEE/IEEE Frontiers in Education Conference. − 2002. − P. 16−21.
13. **Topaloglu N.** Microprocessors and Assembly Language (in Turkish). – Ankara: Seckin Publisher, 2001.
14. **VISUAL 6502** Microprocessor Simulator Home Page. Accessed at: w3.gazi.edu.tr/web/nurettin/visual.

Received 2009 10 10

N. Topaloglu, O. Gürdal. A Highly Interactive PC based Simulator Tool for Teaching Microprocessor Architecture and Assembly Language Programming // Electronics and Electrical Engineering. – Kaunas: Technologija, 2010. – No. 2(98). – P. 53–58.

Teaching microprocessor programming in computing science is one of the challenging tasks of the instructors. This is mainly because of totally new and different subject that needs to be thought to the students. One other subject that produces even more difficulties is making the students to comprehend the I/O operations. There are several hardware platforms especially designed to focus on this matter. There are also several simulation platforms developed as an aid to teach Assembly Language Programming (ALP). In this study we have developed a platform putting extra focus on the I/O aspects of ALP. Our platform provides the user, a highly interactive platform consisting of an editor, a simulator and an animator including several virtual I/O elements. Using the simulator has improved the understanding of the subjects at The Department of Computer Education of Technical Education Faculty of the University of Gazi in Ankara, Turkey. Ill. 7, bibl. 14 (in English; summaries in English, Russian and Lithuanian).

Н. Топалоглу, О. Гурдал. Интерактивный тренажер на основе ПК как средство обучения микропроцессорной архитектуры и языка программирования Ассемблер // Электроника и электротехника. – Каунас: Технология, 2010. – № 2(98). – С. 53–58.

Преподавание программирования микропроцессоров в компьютерной науке является одной из самых сложных задач для инструкторов. Главная причина того это совершенно новая тематика, которую студенты должны освоить. Еще один вопрос, который создает даже больше трудностей, – понятие операций ввода и вывода. Есть несколько аппаратных платформ, специально предназначенных для решения этого вопроса. Есть также несколько платформ моделирования для учения языка программирования Ассемблера (ALP). Мы разработали платформу позволяющую дополнительно сосредоточиться на I/O аспектах ALP. Наша платформа предоставляет пользователю высокую интерактивную среду, состоящую из редактора, симулятора и аниматора, в том числе несколько виртуальных элементов I/O. Использование тренажера улучшило понимание предметов среди студентов департамента компьютерного образования факультета технического образования университета Гази в Анкаре (Турция). Ил. 7, библ. 14 (на английском языке; рефераты на английском, русском и литовском яз.).

N. Topaloglu, O. Gürdal. Interaktyvus kompiuterinis imitatorius, skirtas mikroprocesorių architektūrai ir asemblerio programavimo kalbai mokyti // Elektronika ir elektrotechnika. – Kaunas: Technologija, 2010. – No. 2(98). – P. 53–58.

Mikroprocesorių programavimo mokymas yra vienas didžiausių iššūkių informatikos dėstytojams. Dalykas studentams paprastai būna visiškai naujas ir labai skiriasi nuo kitų paskaitų. Studentams taip pat sunku suprasti įvesties ir išvesties operacijas. Šiam darbui palengvinti skirtos kelios aparatinės platformos. Taip pat yra imitavimo platformų, kuriomis mokoma programuoti asembleriu. Sukurta platforma, kuria naudojantis daugiau dėmesio skiriama įvesties ir išvesties aspektams. Ji suteikia vartotojui galimybę naudotis interaktyviuoju teksto redaktoriumi, imitatoriumi ir animacijos posistemiu. Taip pat numatyti keli virtualūs įvesties ir išvesties elementai. Imitatorius padėjo studentams lengviau suprasti dėstomą medžiagą. Il. 7, bibl. 14 (anglų kalba; santraukos anglų, rusų ir lietuvių k.).