Int. J. Engng Ed. Vol. 26, No. 4, pp. 930–937, 2010
Printed in Great Britain.

0949-149X/91 $3.00+0.00
© 2010 TEMPUS Publications.

SIMAS: A Web-Based Computer System Simulator*

NENAD JOVANOVIĆ
Business School of Professional Studies, Blace, Serbia. E-mail: dzivkovic@singidunum.ac.rs

DRAGAN MARKOVIĆ, DEJAN ŽIVKOVIĆ and RANKO POPOVIĆ
Faculty of Informatics and Management, Singidunum University, Belgrade, Serbia

This paper describes the web-based educational computer system simulator SIMAS. The simulator allows one to enter and compile an assembly language program, as well as load and execute the resulting machine code. Machine instructions can be executed one at a time or in continuous mode. Currently, the executed instruction is highlighted in the computer memory and every phase of its execution is presented visually to the user. The simulator enables the user to visualize the whole execution process of an instruction and to compare assembly and machine language instructions side-by-side. The software package is written in Java as an applet and thus can be run from within any Java-enabled web browser.

Keywords: simulation; visualization; computer system organization; distance learning

1. INTRODUCTION

A WEB-BASED educational computer system simulator, conveniently named SIMAS [1] is described in this paper. The simulator uses animation and visualization to express the process of machine code execution. The software package enables the user to enter an assembly language program and to compile it. The user can then load the resulting machine code into the computer memory and execute it. During code execution, the contents of the processor registers and memory locations are displayed as they change.

Computer system simulators have been widely used for educational purposes in computer systems organization and architecture classes as an instructional aid and a learning tool [11–13]. A representative sample includes ED21 [2], Very Simple CPU [3], DigLC2 [4], CPU Sim [5], LMC Simulator [6], Easy CPU [7], RSIM [8], Logisim [9] and EDCOMP[10].

- ED21 is an educational computer system with a simple but adequate programming language EL21 that visually represents how a program written in high-level programming language translates into assembly language.
- Very Simple CPU uses animation to give students a more intuitive understanding of how the CPU fetches, decodes, and executes instructions. It shows the flow of data within the CPU's register section and ALU.
- DigLC2 is a gate-level simulator that provides a detailed description of all processor components.
- CPU Sim is a computer simulation package that allows the user to specify the processor details.
- LMC Simulator performs tasks by following simple instructions that are described by three-digit numbers. The LMC instruction set is fundamentally similar to the instruction sets of many different computers.
- Easy CPU is a simplified instruction set Intel 80X86 simulator with an animated presentation of the internal computer operations. Easy CPU is developed with the primary objective of enabling a user to become acquainted with the functioning of a computer system. The simulators mainly cover the operations of the processor and show how the instruction fetch and decode operand address calculation and operand fetch, and the operation execution phases are performed.
- RSIM simulates concrete systems of great complexity. The Rsim processor also supports the visual instruction set extensions to the Sparc V9 architecture. The processor supports all instructions except blocked loads and stores and the array instruction.
- Logisim is an educational tool for designing and simulating digital logic circuits.
- EDCOMP supports the animation of instruction execution and allows students to write their own assembly programs, translate them, interactively set and examine values of memory locations, registers, and input/output units, and run simulation.

All of these simulators use a graphical user interface and interactive mode of operation to actively engage users in the simulation process. However, there are differences between them in the simulation approach and thus in their features.

* Accepted 12 November 2009.

The most notable differences are as follows.

- Some simulators support only fixed processor architecture, while others allow users to design their own relatively simple architecture.
- Depending on the simulated target system, simulation is carried out on the instruction or clock level.
- Some simulators can be run from within any web browser and so they are even suitable for distance learning classes. Other simulators must be installed and run only locally.

Table 1 summarizes the features of the above mentioned simulators according to these criteria.

The primary goal of the SIMAS simulator is to enhance the learning of computer system organization and architecture. It visually shows how a program-controlled processor works, based on animated simulation. It also aims to enable students to gain a better understanding of all the steps that are necessary in the development of an assembly language program and in the execution of a machine language instruction. SIMAS is a simple and effective learning environment for students who are studying computer system organization and operation.

The software package is written in Java as an applet and can thus be run from within any Java-enabled web browser. Various versions of this tool have been used as a teaching aid for five years.

The novel aspect of the use of SIMAS in education is pedagogical: it is based on experience and the analysis of some factors such as educational level, learning style, systematic work, expectations of the students, etc. the simulator is adapted to the three-year professional studies with an emphasis on practical work.

There are three levels of students (fair, good, excellent) that are grouped after the evaluation. The learning material is adapted for a particular group. Depending on the style of learning, three groups of students were formed (visualists, auditory and practical). Once again the learning material is adapted for a particular group. Our starting point is the active role of students during the learning process. We have compared the results of students who were learning at a distance and those taught in the traditional way. The aim was not to show the advantages and disadvantages of different systems of education: the same teachers prepared the learning materials and the differences were not large. We used the experiences of the students and professors working in mutual cooperation and exploration.

The SIMAS assessment strongly suggests that it is an effective learning tool for all students, even students with different learning characteristics.

2. SIMULATED COMPUTER SYSTEM OVERVIEW

Computer system simulated by SIMAS contains CPU, the main memory, and input/output devices.

The main memory stores input data, auxiliary data, output data, and program instructions. The main memory consists of a large number of addressable locations in which binary values can be saved. Each memory location has a fixed length of n bits and thus can store any of the 2^n binary values.

Memory locations are indexed by integers 0 to (m—1), which are used to efficiently access the contents of a memory location. The index of a memory location is its address, and the total number of locations, m, represents the memory capacity.

The CPU contains the arithmetic/logic unit (ALU), control unit, and fast registers. The fast registers locally store internal CPU data that are acted upon by ALU performing arithmetic and logic operations. The CPU registers are divided into two groups: addressable and internal. The addressable registers have their addresses and the programmer can use them to write and read data, whereas internal registers are 'hidden' from the programmer and serve for internal CPU operations.

The ALU performs arithmetic and logic operations. It gets an operation's operands from the CPU registers and puts results back into them.

The control unit manages other components of the system and coordinates their operation. In particular, it

- directs reading from and writing to the main memory;
- directs data transfer between the ALU and the main memory;

Table 1. Characteristics of the representative simulators

Simulator	System architecture	Simulation mode level	Web
Ed21	Fixed	Instruction	Yes
Very Simple CPU	Fixed	Instruction	Yes
Diglc2	User-defined	Instruction	No
Cpu sim	User-defined	Clock	No
Lmc	Fixed	Instruction	Yes
Easy cpu	Fixed	Clock	Yes
Rsim	Fixed	Clock	No
Logisim	User-defined	Clock	No
Edcomp	User-defined	Clock	Yes
Simas	Fixed	Instruction	Yes

- synchronizes the operations of other elements of the system.

The control unit performs these functions by repeatedly fetching program instructions from the memory and decoding them. Based on the decoded type of the instructions, it then generates corresponding control signals to execute the instructions.

During system operation the CPU communicates with the main memory, which stores program data and instructions. The instructions tell the CPU which operations should be performed and on which data.

The various parts of the computer system are connected by buses, which are sets of conductors that transfer multiple bits in parallel. The transfer of data and instructions between the CPU and the memory is along the bidirectional data bus. It is connected to an internal CPU register called the memory data (MD) register. The memory location to which the CPU writes data or reads data from is determined by the address stored in another internal CPU register called the memory address (MA) register.

The MA register is connected to the unidirectional address bus through the address decoder. The address bus automatically references the memory location whose address is stored in the MA register. The width of the address bus determines the maximum memory capacity. For example, if the width is m bits, then the capacity of the main memory maybe, at most, 2^m locations.

Another bus that is internal to the CPU, called the control signal bus, transmits control signals from the control unit to the ALU. These control signals direct the ALU to perform a particular operation, such as addition.

The CPU contains two more internal registers. The instruction register (IR) stores the instruction fetched from the main memory that is about to be executed. The buffer register (BR) temporarily stores arguments and intermediate results of the operations.

The addressable registers in the CPU have **specific** functions and their number depends on particular processor architecture. Nevertheless, common addressable registers found in almost all CPUs are:

- a Program Counter (PC), which contains the memory address of the next instruction that should be fetched and executed;
- a Stack Pointer (SP), which that contains the memory address of the program stack;
- a Status Register (SR), which contains information about the current state of the CPU and the program being executed; and
- an Accumulator (AC), which stores some value to which can be added a given value and the result put back in the accumulator.

2.1 Machine instruction execution

In general, the execution of a machine instruction by the CPU is made up of the following steps.

1. Fetching the instruction from the main memory.
2. Decoding the instruction in the CPU.
3. Determining the address of the instruction's operands.
4. Executing the specified operation on the operands.
5. Storing the results.

Computer architecture based on the program counter that contains the address of the next instruction to be executed is called *von Neumann architecture*. The CPU operation in this type of computer architecture is divided into two phases that are carried out continuously:

Phase 1: Fetching the instruction from the main memory and readying it for execution

- The address of the instruction to be executed is contained in the program counter register (PC). This address is moved to the memory address register (MA), which causes a memory request to be sent along the memory bus for reading out the contents of the corresponding memory location.
- Once a control signal is sent to read out the contents of memory location whose address is in the MA register, the contents is moved along the data bus into the memory data register (MD), and then into the instruction register (IR).
- The instruction in the IR register is analyzed and, if it is found to be incorrect, the execution of the program is interrupted.
- The contents of the program counter (PC) is increased by the length of the current instruction (in memory words), so that the PC now contains the address of the next program instruction in memory.

Phase 2: Execution of the instruction

- The addresses of operands are determined.
- The operands are fetched from memory.
- The instruction code is decoded and the ALU performs the corresponding operation.
- The results are stored in memory, if necessary.

3. SIMULATED COMPUTER SYSTEM DETAILS

The hypothetical computer is chosen so that it is powerful enough to express main ideas, but not overly complex to blur the general picture. We postulate that it is a 16-bit, single-processor, single-task computer. This means it contains one CPU based on the von Neumann architectural model. Also, the length of each machine instruction, memory word, or CPU register is 16 bits. The size of the main memory is $2^{16} = 65536$ memory locations.

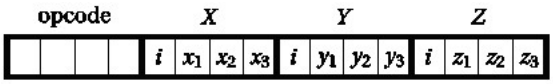

Fig. 1. The machine instruction format.

Table 2. The machine instruction set

Instruction	Opcode	Mnemonic
Data transfer	0000	MOV
Addition	0001, 1001	ADD
Subtraction	0010, 1010	SUB
Multiplication	0011, 1011	MUL
Division	0100, 1100	DIV
Branch if equal	0101	BEQ
Branch if greater	0110	BGT
Input data	0111	IN
Output data	1000	OUT
Program halt	1111	STOP

3.1 Machine instruction format

Each machine instruction has a fixed format made up of four fields: operation code and three operand addresses. This is illustrated in Fig. 1.

If the address of an operand uses 4 bits, we can directly address only $2^4 = 16$ memory locations. However, the first bit of the operand address indicates whether the address is real or indirect. That is, if $i = 0$, the next three bits determine a real address; if $i = 1$, the next three bits determine the address of memory location that contains the real operand address. In this way we can address all of the 65536 memory locations.

The 4-bit operation code field (opcode) allows for 16 different instructions. The instruction set for the CPU contains ten instructions. The instructions are chosen to represent instruction types commonly found in systems of this level:

- data transfer operation (opcode 0000)
- integer arithmetic operations (opcodes 0001–0100 and 1001–1100)
- branching operations (opcodes 0101 and 0110)
- input/output operations (opcodes 0111 and 1000)
- program halt (opcode 1111)

3.2 Assembly language instructions

Writing programs in machine language is prone to error and is time-consuming because the programmer must remember numeric operation codes and calculate addresses by hand.

Table 2 lists the mnemonics used for the operation codes of the machine instruction set.

4. USING SIMAS

The SIMAS simulator is very simple to use. It can be started from within any Java-enabled web browser. The opening window (Fig. 2) displays simulator controls and the simulated system.

The user can specify an assembly language program by using an editor or by getting its source code from a text file. The editor is started by choosing the 'Source'button, and the source file

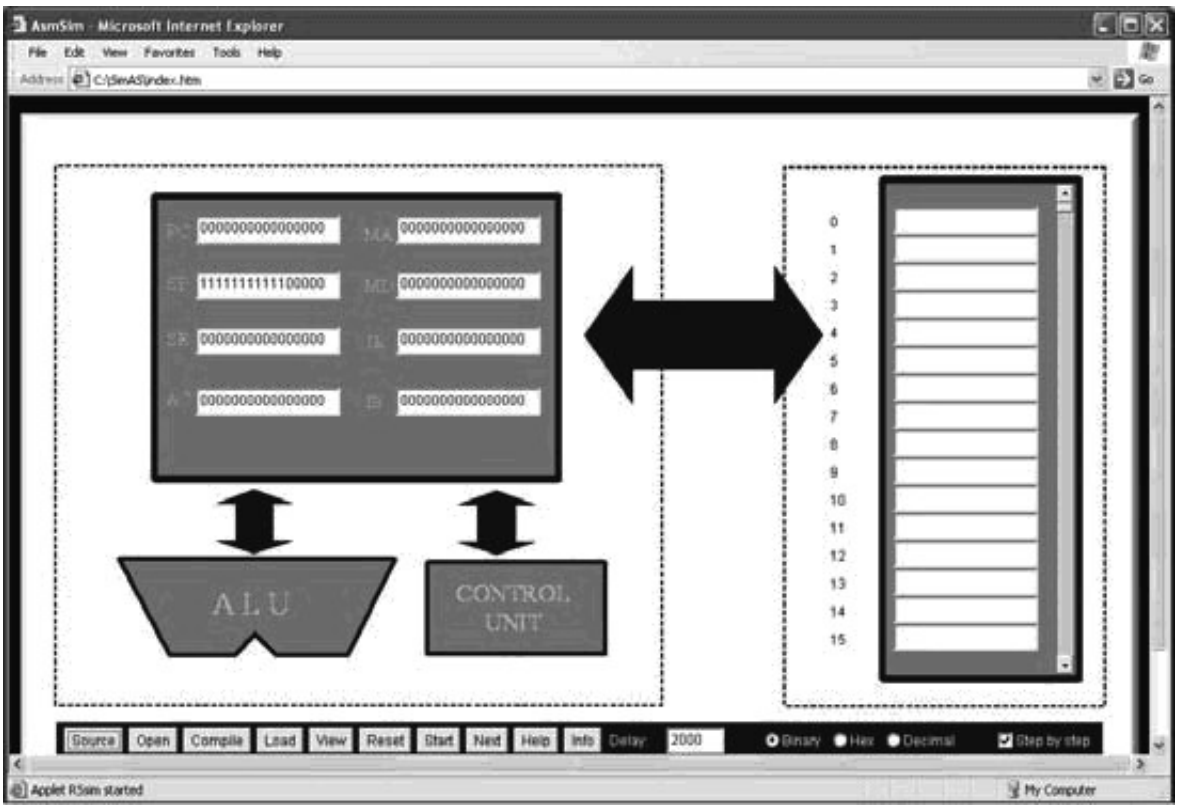

Fig. 2. The simulator's main window.

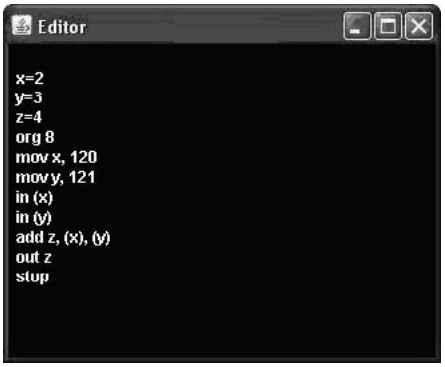

Fig. 3. The simulator's editor.

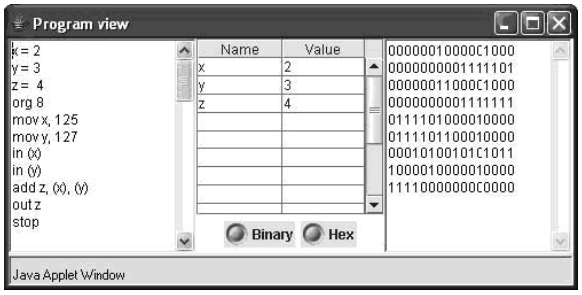

Fig. 4. The simulator's view window.

is selected by choosing the 'Open'button. Figure 3 shows the simulator editor's window.

The source code is compiled into machine instructions by choosing the 'Compile' button. The button View opens a window (Fig. 4) with the left part showing the source code of the assembly language program, the middle part showing the symbol values, and the right part showing the assembled machine language program. Depending on the selected radio button, 'Binary' or 'Hex', the machine code in the right part of the program view window is displayed in the binary or hexadecimal number system.

After successfully assembling a program, the user can load the machine code into the main memory by choosing the 'Load' button.

The animation of the execution of a loaded program is started by choosing the 'Start' button. The machine code instructions can be executed by single stepping through each instruction ('Step by Step' checked), or in continuous mode with a delay between execution of two instructions given in the 'Delay' field. If the execution of one instruction at a time is selected, the next instruction is executed by choosing the 'Next' button. The simulator marks the instruction currently being executed by highlighting it in the main memory.

4.1 An example of the use of SIMAS

To illustrate the use of SIMAS as a learning tool in assembly language programming, consider the assembly language program in Figure 3. For instance, the instruction:

```
mov x, 125
```

has two arguments. The first argument is the symbol x whose value 2 is looked up in the symbol table. The second argument is the literal value 125. The instruction thus stores the value 125 into the memory location whose address is 2.

This assembly language instruction is assembled into a machine instruction occupying two memory words:

```
0000  0010  0000  1000
0000  0000  0111  1101
```

The first memory word defines opcode 0000 for the MOV operation, binary value 0010 for the decimal value 2 of the symbol x and indirect addressing mode 1000 for the value contained in the next memory word (0000 0000 0111 11012 = 12510). The first argument of the instruction is directly addressed because the first bit of its address field is 0. The remaining three bits, 010, then determine the address of the first argument.

Figure 5 shows the contents of the memory and CPU registers before the instruction is executed. The program counter contains the address (8) of the next instruction to be executed; the instruction register contains that very instruction, and the memory address register contains the address of the memory location that receives the value contained in the memory data register.

Other instructions in the example assembly language program can be analyzed in a similar way.

5. SIMAS ASSESSMENT

SIMAS has been used for the last five year in our web lab. The participants were undergraduate students from the Computing and Informatics Department of Business School of Professional Studies, Blace, Serbia.

We have conducted a qualitative and quantitative evaluation of the use of SIMAS in the classroom. The qualitative evaluation included a number of student surveys and discussions with fellow instructors who teach courses that directly or indirectly include a course using SIMAS as a prerequisite. The surveys have been aimed at learning what students perceived to be a good educational tool and how they assessed the overall effectiveness of this approach. The majority of students praised the graphical representation and found it user friendly. Fellow instructors have reported that students using SIMAS were better prepared and had a deeper understanding of the basic concepts. The environment is interactive and is applicable in laboratory exercises. Each exercise has four components: prelab preparation, in-lab knowledge assessment, in-lab assignment, and a written report. To prepare for a particular lab, the students must review related material from lectures and the textbook, and read the related sections from the lab manual. Students can interact with the simulators asynchronously at any time for as

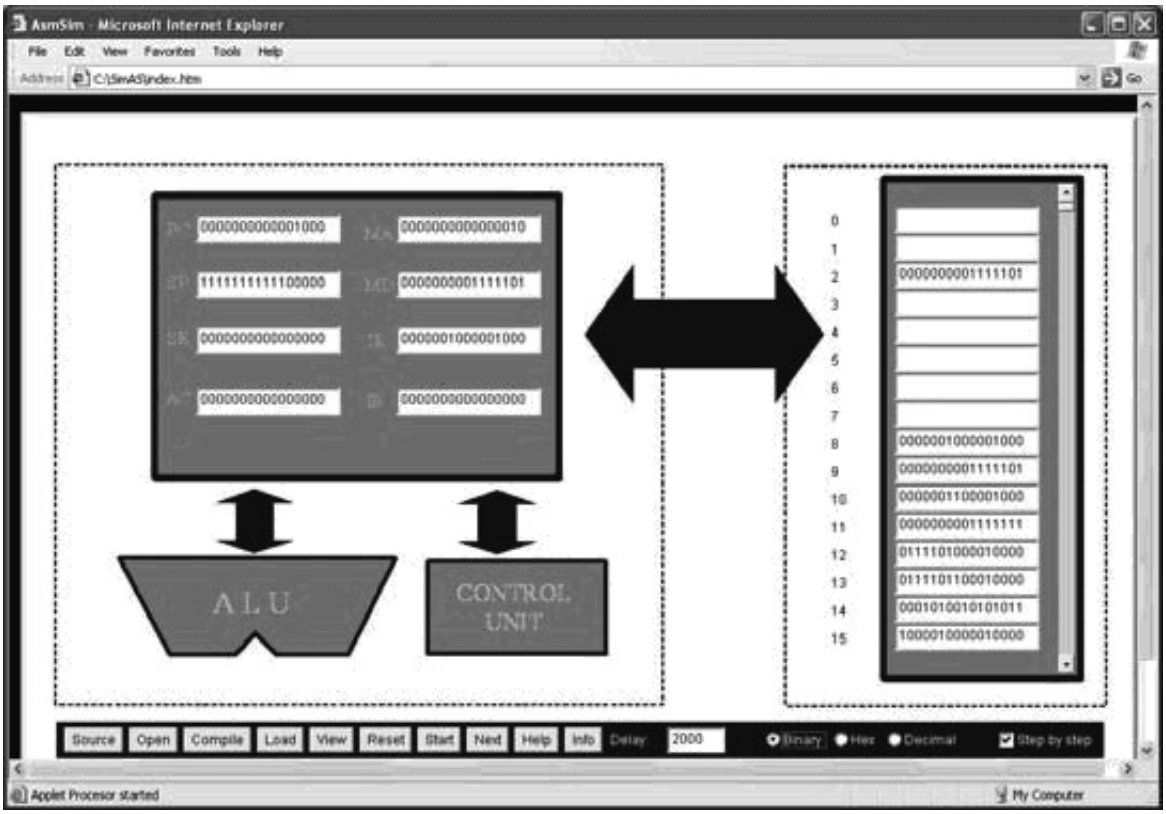

Fig. 5. A snapshot of the memory and CPU.

long they want. Each lab assignment is preceded by a short computer-based test designed to verify whether the students understand the topic covered in the assignment. After passing the test, the students select a predefined experiment. Based on the observations made during the experiment, they answer questions relevant to the topic and turn in a written report.

More importantly, the quantitative evaluation substantiated this subjective perception: the percentage of students passing the exam increased steadily from below 50% before 2005 when we started using SIMAS (Fig. 6).

Starting assumptions and objectives are based on pedagogical principles, experience and analysis of some parameters such as level of education, learning style, systematic work, the expectations of the students, etc.

Learning styles represent different approaches or ways of studying. Every student, when acquiring knowledge, takes in the information that he or she receives through a certain modality and therefore, by using that information, learns in the most efficient way. According to that modality, the basic typology of learning styles is as follows: visual, auditory and tactile/ (practical activity) learning styles. Visual learning is dominant in those who acquire information when it is presented visually in the form of text (graphical–visual) or a picture. They mostly prefer individual learning. Those who find it easier to learn by listening to lectures, discussions, exchanges of ideas, use the auditory learning style. This is the reason why it is characteristic of this learning style for students to work in pairs or groups. Those who take notes, draw pictures and diagrams during the learning process, in order to memorize the information more easily use the tactile/kinesthetic learning style. They learn best through movement, games, acting or concrete action, actively exploring the physical world around them.

According to the previously mentioned learning styles, we have tested a group of students (over 300 students) and arrived at the following results.

First, a preliminary test was performed. This was carried out in order to determine the particular learning styles of students and the best way to express knowledge.

The results of the test were as follows: 24% of

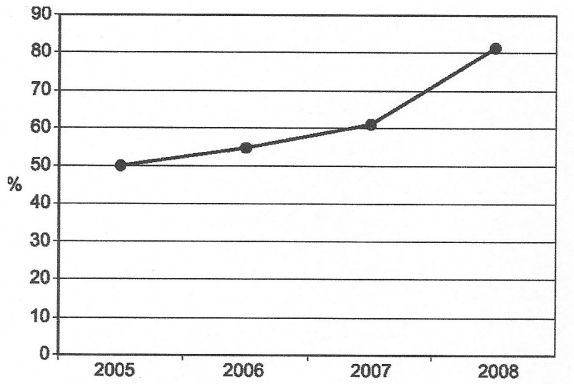

Fig. 6. Percentage of students who passed the exam on the first try.

the students prefer to study based on seeing—the visualists, 36% prefer to study by listening—the auditory and 40% of the students prefer to study by practical activity.

Based on this, the students were divided into three groups: visualists, auditory and practical.

Testing showed that the system described in this paper proved to be accessible to the three groups of students.

6. CONCLUSIONS

We have presented a web-based educational environment whose main goal is to be an effective instructional and tool in learning assembly language programming. We strongly believe that the visualization it offers enhances the student's understanding of the basic logical structure of a general computer system. By animating the execution phases in the CPU, the simulator provides students with a more intuitive understanding of how the CPU fetches, decodes, and executes instructions. The SIMAS simulator is written in Java as an applet so that it can be run from within any Java-enabled web browser, which makes it suitable even for distance learning classes.

We can conclude that:

- simulation is increasingly being used as a tool to support the teaching of assembly language programming and computer organization;
- when simulators are combined with visualizations they become an even more effective teaching tool.

We believe in the potential of visualization to improve the teaching process. This has motivated us to share our experiences. We plan to continue this collaboration with the aim of better designing evaluation methods and formulating appropriate visualization concepts. This project is not fully complete, as several other features are being designed or under development, such as new assessment tool, students profiles, etc.

The positive experience of the educational model based on the SIMAS simulator applied in a computer system course led us to adapt the simulator model to other courses that we also teach.

REFERENCES

1. SIMAS demo version: http://weblab.vpskp.edu.rs/osovi/simas/index.htm
2. Y. Miura, K. Kanekoand and M. Nakagawa, Development of an educational computer system simulator equipped with a compilation browser, *International Conference on Computers in Education*, 2004.
3. J. D. Carpinelli, The very simple CPU simulator, *32nd ASEE/IEEE Frontiers in Education Conference*, Boston MA, USA, 2002.
4. A. Cohen and O. Temam, Digital LC-2: From bits & gates to a little computer, *WCAE*, Alaska, 2002.
5. D. Skrien, CPU Sim 3.1: A tool for simulating computer architectures for computer organization classes, *ACM JERIC*, 2001, pp. 46–59.
6. W. Yurcik and L. Brumbaugh, A web-based little man computer simulator, *Technical Symposium on Computer Science Education (SIGCSE)*, Charlotte NC, USA, 2001, pp. 204–208.
7. C. Yehezkel, W. Yurcik and M. Pearson, Teaching computer architecture with a computer-aided learning environment: state-of-the-art simulators, *Proc. 2001 Intl. Conf. on Simulation and Multimedia in Engineering Education (ICSEE)*, Phoenix AZ, USA.
8. C. Hughes, V. Pai, P. Ranganathan and S. Adve, RSIM: Simulating Shared-Memory Multiprocessors with ILP processors, *IEEE Computer*, **35**(2), 2002, pp. 40–49.
9. C. Burch, Logisim: A graphical system for logic circuit design and simulation, *Journal of Educational Resources in Computing*, **2**(1), 2002, pp. 5–16.
10. J. Djordjevic, B. Nikolic and A. Milenkovic, Flexible web-based educational system for teaching computer architecture and organization, *IEEE Transactions on Education*, **48**(2), May 2005, pp. 264–273.
11. Y. Imai, K. Kaneko and M. Nakagawa, A web-based visual simulator with communication support and its application to computer architecture education, *Seventh IEEE International Conference on Advanced Learning Technologies (ICALT 2007)*, IEEE, 2007.
12. K. Nakano and Y. Ito, Processor, assembler, and compiler design education using an FPGA, *14th IEEE International Conference on Parallel and Distributed Systems*, 2008.
13. H. Yanagisawa, M. Uehara and H. Mori, Evaluation of automatic generation of an instruction set simulator for educational use, *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW06)*, IEEE, 2006.

Nenad Jovanović received his M.S. and Ph.D. degrees in electrical engineering from the University of Priština, Serbia. He is currently an Assistant Professor with the Department of Computer Engineering, School of Professional Studies, Blace, Serbia. His research interests include operating systems, computer networks, distributed systems, programming languages, multimedia and distance learning.

Dragan Marković received his M.S. and Ph.D. degrees in electrical engineering from the University of Belgrade, Serbia. He is currently an Assistant Professor with the Department

of Computing and Informatics, Faculty of Informatics and Management, Singidunum University, Belgrade, Serbia. His research interests include digital image processing, user interfaces, multimedia, and information retrieval.

Dejan Živković received his M.S. degree in computer science from the University of Belgrade, Serbia and his Ph.D. in computer science from the Wesleyan University, USA. He is currently an Associated Professor with the Department of Computing and Informatics, Faculty of Informatics and Management, Singidunum University, Belgrade, Serbia. His research interests include complexity theory, algorithms and programming systems.

Ranko Popović received his M.S. and Ph.D. degrees in electrical engineering from the University of Priština, Serbia. He is currently an Associated Professor with the Department of Computing and Informatics, Faculty of Informatics and Management, Singidunum University, Belgrade, Serbia.