Flexible Web-Based Educational System for Teaching Computer Architecture and Organization

Jovan Djordjevic, Bosko Nikolic, and Aleksandar Milenkovic, *Member, IEEE*

***Abstract*—An important problem in teaching courses in computer architecture and organization is to find a way to help students to make a cognitive leap from the blackboard description of a computer system to its utilization as a programmable device. Computer simulators developed to tackle this problem vary in scope, target architecture, user interface, and support for distance learning. Usually, they include the processor only, lacking the whole-system perspective. The existing simulators mainly focus on the programmer's view of the machine and do not provide the designer's perspective. This paper presents an educational computer system and its Web-based simulator, designed to help teaching and learning computer architecture and organization courses. The educational computer system is designed to cover a broad spectrum of topics taught in lower division courses. It offers a unique environment that exposes students to both the programmer and the designer's perspective of the computer system. The Web-based simulator features an interactive animation of program execution and allows students to navigate through different levels of the educational computer system's hierarchy—starting from the top level with block representation down to the implementation level with standard sequential and combinational logic blocks.**

***Index Terms*—Computer-aided instruction, computer architecture, computer science education, educational technology, simulation.**

I. Introduction

COMPUTER architecture and organization courses play a central role in the computer engineering curriculum. They provide students with knowledge in all aspects of the design and organization of the central processing unit, memory, input/output (I/O) subsystem, and their incorporation into a computer system. While computer architecture encompasses the programmer's or abstract view of the machine, computer organization deals with implementation details. However, these two views cannot be treated separately because of their strong mutual dependence [1]. The joint IEEE Computer Society and ACM Computer Engineering Task Force has identified the core topics in computer architecture and organization [2], shown in Table I. Computer engineering graduates should have a very good theoretical and practical understanding of these topics.

The laboratory where students get hands-on experience is crucial in helping the students to bridge the gap between theoretical knowledge and practical problems architects face in designing various computer system modules. The most common approach in narrowing this gap relies on software simulators of computer systems. Software simulators have several advantages over "real" microcomputer platforms; they are less expensive, more flexible, and more appropriate for lower division courses, which typically include a large number of students. In addition, graphical presentation and animation help students to "experience" computer system functioning and better understand various design issues.

This paper presents EDCOMP, an educational computer system and its Web-based simulator. The computer system demonstrates a broad spectrum of topics typically taught in lower division computer architecture courses; the topics supported are in italic type in Table I. The Web-based graphical simulator supports 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 I/O units; and run simulation. It gives a visual presentation of all parts of the computer system both at the level of standard system modules and at the level of combinational and sequential circuits; displays values of signals; performs simulation at the level of a clock cycle, an instruction, and a complete program; and displays timing diagrams of selected signals. The simulator flexibility allows instructors to focus on a specific topic. For example, when studying integer multiply instruction execution, students can run a simulation on a clock-by-clock cycle basis, closely following the changes in the relevant registers and arithmetic and logic unit (ALU). On the other hand, when learning I/O fundamentals, students can follow simulation at the system level, where the processor is considered as a black box, executing a program on the instruction-by-instruction level.

The paper is organized as follows. Section II provides background and motivation. Section III introduces the structure of the educational computer system and its components, and Section IV concentrates on the visual simulator. Section V describes laboratory organization, and Section VI illustrates the capabilities of the educational environment through several laboratory experiments. Section VII describes the evaluation of the educational environment, and Section VIII concludes the paper.

II. Background and Motivation

The course Introduction to Computer Architecture and Organization at the School of Electrical Engineering, University of Belgrade, Belgrade, Serbia and Montenegro, is mandatory for students with communications, automation and electronics, and computer engineering majors. This course is taught in the

Manuscript received January 21, 2004; revised August 7, 2004.

J. Djordjevic and B. Nikolic are with the School of Electrical Engineering, University of Belgrade, 11000 Belgrade, Serbia and Montenegro (e-mail: jdjordjevic@etf.bg.ac.yu).

A. Milenkovic is with the Electrical and Computer Engineering Department, The University of Alabama, AL 35899 USA.

Digital Object Identifier 10.1109/TE.2004.842918

TABLE I
CORE TOPICS IN COMPUTER ARCHITECTURE AND ORGANIZATION

Fundamentals	Organization of the CPU	Computer Arithmetic	Main Memory	Interfacing and Communication
• *Registers and register file* • *Data types* • *Instruction types* • *Addressing modes* • *Instruction formats* • *Fetch, decode, execution cycles* • *I/O techniques and interrupt*	• Single vs. multiple bus datapaths • Pipelined, non-pipelined • *Control unit: hardwired vs. microprogrammed realization* • *Arithmetic units implementation*	• *Representation of integers (signed, unsigned)* • *Basic arithmetic algorithms for integer addition, subtraction, multiplication, and division* • *Representation of real numbers* • *Basic arithmetic algorithms for operations on real numbers* • *Conversions between real and integer numbers*	• *Memory hierarchies* • *Main memory organization* • *Latency, bandwidth, cycle time, performance* • Virtual memory system • Cache memories • Memory interleaving • Memory technologies (SRAM, DRAM, EPROM, Flash) • Reliability and error correction	• *I/O fundamentals: handshaking, buffering* • *I/O techniques: programmed I/O, interrupt driven, DMA* • *Interrupt structures: vectored and prioritized, interrupt overhead, interrupts and re-entrant code* • *Buses: clock, control, address and data buses, arbitration* • Parallel and serial interfaces • Timers

TABLE II
EDUCATIONAL COMPUTER ARCHITECTURE SIMULATORS: AN OVERVIEW

System	Complexity	Instruction Set	Graphical presentation	Simulation granularity	Simulation Mode	Implement. Details	Distance Learning
HASE [6]	High	User	Yes	Clock	Interactive	No	Yes
ASF [7]	Medium	User	No	Program	Batch	No	No
ESCAPE [8]	Low	Comm.	Yes	Clock	Interactive	Yes	Yes
RM [9]	Low	Comm.	Yes	Clock	Interactive	Yes	No
SimpleCPU [10]	Low	Comm.	Yes	Clock	Batch	Yes	No
Easy CPU [11]	Low	Comm.	Yes	Clock	Interactive	No	Yes
ANT [12]	Medium	Comm.	No	Instruction	Batch	No	No
Newsport [13]	Medium	Comm.	No	Clock	Interactive	No	No
CASLE [14]	Medium	Comm.	No	Instruction	Batch	Yes	Yes
SimpleScalar [3]	High	Comm.	No	Clock	Batch	No	No
DLXview [5]	High	Comm.	Yes	Clock	Interactive	Yes	No
RSIM [15]	High	Comm.	No	Clock	Batch	No	No
SimOS [4]	High	Comm.	No	Program	Batch	No	Yes

second year, and its prerequisites are first-year courses Introduction in Digital Logic and Programming Languages. The goal of this course is to give a broad overview of computer architecture and organization topics. For computer engineering students, this course is followed by Advanced Computer Architecture, Microprocessors and Interfacing, Computer VLSI (very large scale integration) Design, and Parallel Processing. The course includes a mandatory laboratory component based on the use of a software simulator of a computer system.

A variety of educational simulators designed to support teaching courses in computer architecture and organization have been proposed and developed. Table II gives an overview of educational simulators and their characteristics. They differ greatly in scope and complexity (rudimentary, medium, complex), type of instruction set (commercial, custom), user interface, simulation granularity (program, instruction, clock), simulation mode (batch, interactive), level of implementational details, and support for distance learning. For instance, SimpleScalar [3] and SimOS [4] represent highly sophisticated simulation suites widely used in computer architecture research. However, high complexity and a lack of graphical interfaces make them impractical for illustration of the fundamental topics presented in Table I. DLXview [5] is a powerful educational system that supports animation of instruction execution at the clock level, but it does not include implementational details, and it concentrates only on the processor, lacking the whole-system perspective.

Ideally, the simulators should support examples for a wide range of relevant topics, allow students to write their own assembly and/or high-level language programs, and allow simulation and graphic animation of the program execution at the various levels of hierarchy. Graphical representation and animation of computer system operation has proven to be quite a powerful tool in teaching computer architecture [16]. The educational simulators must also be user friendly with a minimal learning curve and configurable, allowing instructors to adapt them to a number of various laboratory experiments and various courses' curricula. Finally, all modern educational

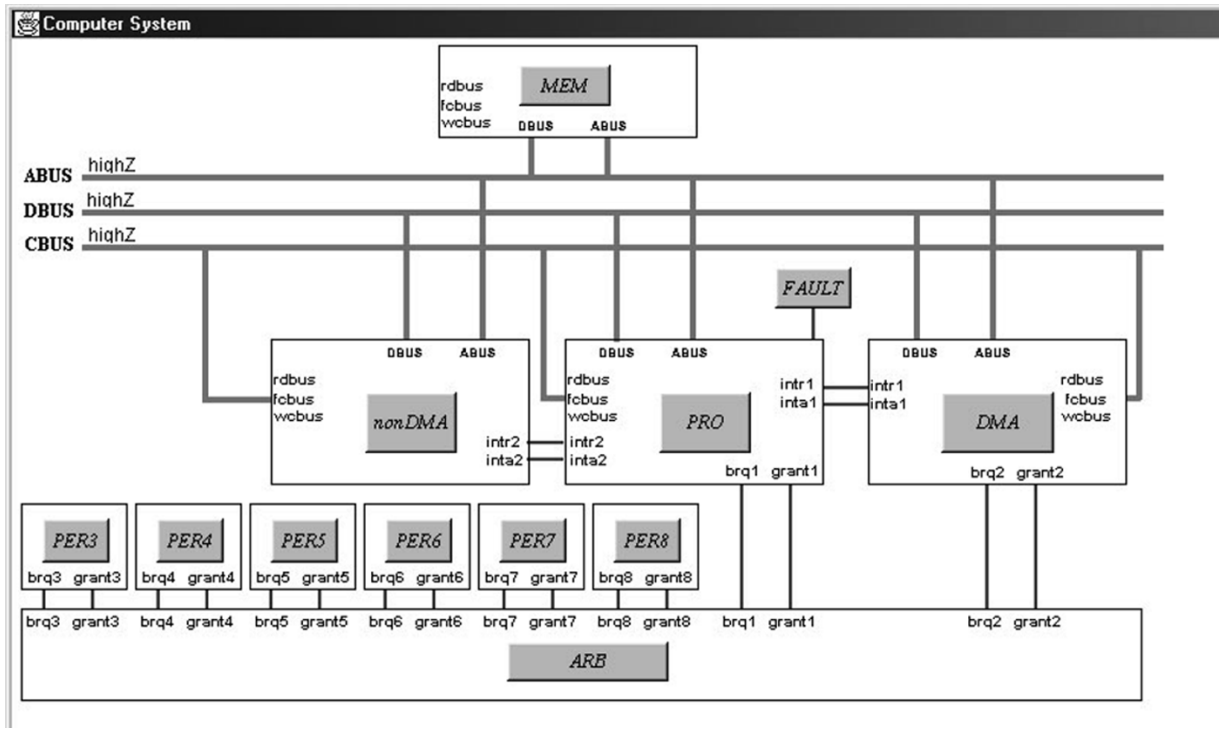

Fig. 1. Structure of EDCOMP.

tools striving for wide acceptance should support distance learning. These requirements are often contradictory, making the development of such a software tool very difficult.

None of the existing educational computer architecture simulators completely satisfied the previously mentioned requirements; thus, the authors decided to develop a new simulation environment with a graphical easy-to-use user interface that offers both a programmer's and a designer's view of a computer system. A number of design choices have been made. First, a computer architecture tailored specifically to educational needs was selected. Although simulators based on commercial architectures provide hands-on experience with real-world systems and can decrease the cost of development, they are rarely ideal choices for educational purposes. Off-the-shelf systems are designed to achieve optimum performance at minimal cost. They do not cover various computer architecture topics in breadth; for example, only a subset of possible addressing modes is supported. Students learn about commercial systems in higher division courses, such as Microprocessors and Interfacing.

Another issue was a choice between complex instruction set computer (CISC) and reduced instruction set computer (RISC) architectures. The authors opted for a CISC processor for the following reasons. The follow-up computer engineering course, Advanced Computer Architecture, deals with RISC architectures, and a similar educational environment is developed around an educational RISC processor with pipelined organization [16]. Second, the nature of RISC processors is to have a limited subset of the most frequently executed instructions, while other instructions are emulated in software. This setup is in direct opposition with the intention to provide an architecture with a rich instruction set, suitable for explanation of arithmetic algorithms, various addressing modes, and interrupt mechanism. Finally, the authors believe that computer engineering and science students should be exposed to both architecture styles, since CISC architectures are still dominant in the desktop and embedded processor market (e.g., x86, 8051, and PICmicro microcontrollers).

The initial version of the simulator was developed as a standalone application in Java. After several years of use, a transition was pursued to a Web-based environment for the following reasons. The large number of students enrolled in the course, in excess of 400, placed tremendous pressure on laboratory equipment and staff. The Web-based environment allows students to prepare for laboratory at home, at their own pace, thus reducing the time needed for successful completion of laboratory exercises. The Web-based technology also offers seamless integration with knowledge assessment and administrative tasks and cost reductions for installations, updates, and maintenance.

III. Architecture and Organization of EDCOMP

EDCOMP includes a processor (PRO), a main memory (MEM), an I/O subsystem with a direct memory access (DMA) controller, a non-DMA controller, and six dummy peripheral controllers (PER3–PER8), and an arbitrator (ARB), as shown in Fig. 1. All components communicate through an asynchronous bus with address, data, and control lines. Each bus master is connected with the bus ARB by a pair of private lines: the bus request line brq_i and the bus grant line grant_i. A dummy controller only takes part in the bus arbitration, keeps the bus busy for a definable period of time, and performs no transfer of data.

The processor features an originally developed CISC architecture. It includes separate data, address, base and index registers, and special-purpose registers, such as the program counter,

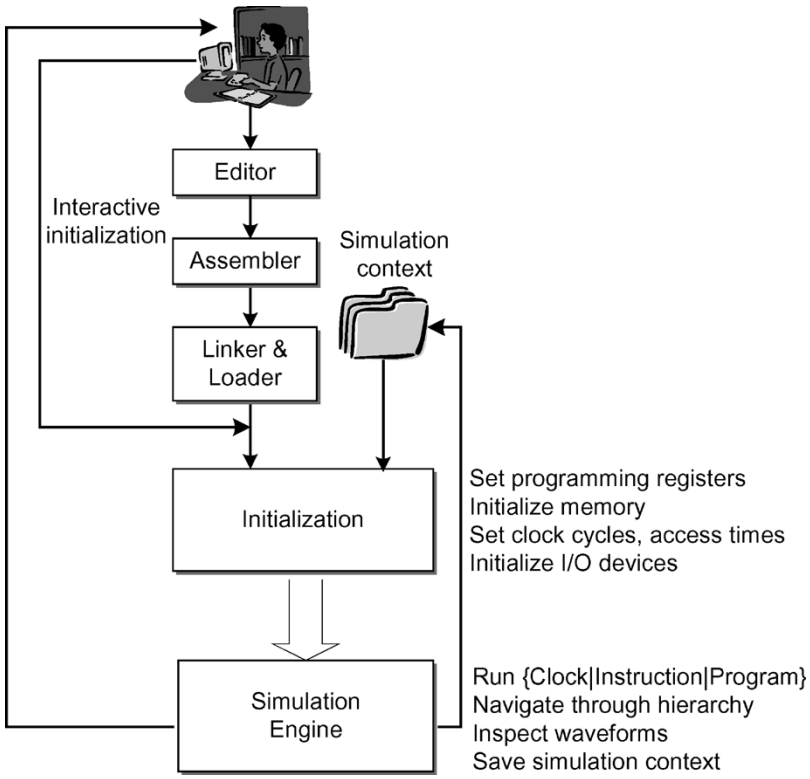

Fig. 2. Simulation flow.

the stack pointer, the program status word, and the accumulator. Data types supported are 8-b signed and unsigned integers, 16-b floating-point numbers, and strings of characters; the size of data types is chosen to provide a relatively modest number of steps for arithmetic algorithms (e.g., shift and multiplication algorithm for integers). The instruction format is one address with a variable instruction length of 1, 2, 3, and 4 B. An extensive set of addressing modes is supported, including register direct, register indirect, memory direct, memory indirect, base, index, base index, base with 8- and 16-b displacement, base index with 8- and 16-b displacement, relative, register indirect with auto increment and auto decrement, and immediate. The instruction set includes the transfer, arithmetic with integer and floating-point data types, logic, shift, rotate, control, loop control, and string instructions. The interrupt mechanism is vectored with internal and external interrupts. The processor has two interrupt request lines for external maskable interrupt requests and one line for a nonmaskable request. The processor features separate units for each of the instruction execution phases: the instruction fetch unit (IF), five operation execution units—integer (IE), floating-point (FPE), string (SE), control (CE), and loop control (LCE)—and the interrupt service unit (IS). All these units use the bus interface unit (BI) for bus arbitration and read/write bus cycles. Each of these units has its own hard-wired control unit (CU). The detailed description of EDCOMP is available in [17].

IV. Visual Simulator Capabilities

The visual simulator is run from a Web browser and includes various menus to initialize and control simulation of EDCOMP. The flow of a simulation session is shown in Fig. 2. The initialization supports an easy preparation of test examples for instructors and an easy initialization of the whole educational system by loading a predefined file for students. The initialization allows the user to define the clock rates for the system modules [Fig. 3(a)] and the access times of the peripheral devices and the memory, examine and set the values of memory locations and registers of the processor [Fig. 3(b)] and I/O units, and select signals for which the timing diagrams will be drawn. Each module can be initialized separately, interactively, or from a file. In addition, the complete initialization from a file is possible. Memory locations can be initialized interactively by setting values directly or by using the editor, the assembler, the linker, and the loader. The result of an initialization or the context of the whole system at any point of time during simulation can be saved in a file.

The simulator graphically presents parts of the computer system and signal values, simulates the behavior of EDCOMP, and displays simulation results in a user friendly manner. During a simulation run, two windows are present on the screen (Fig. 4). The window in the upper part of the screen, named the block diagram window, shows parts of the computer system. Because a limited number of elements can be displayed on a screen, a two-level hierarchical scheme of screens is developed. The first-level screen gives the structure of the computer system at the level of modules (Fig. 1), and the simulation run begins with this screen. For a more detailed structure of any of the modules, the user needs to point and click to select a module to go to the second level of screens. The second-level screen gives the structure of the module's processing unit at the level of combinational and sequential circuits (Fig. 4). The exception

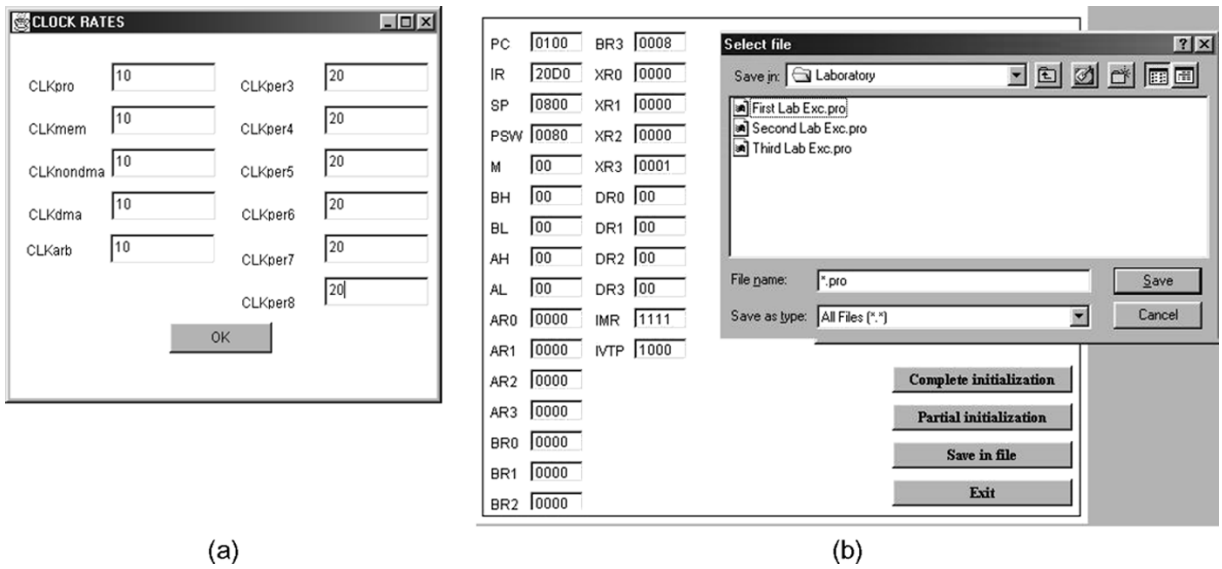

Fig. 3. (a) Initialization of clocks rates. (b) Partial initialization of processor registers.

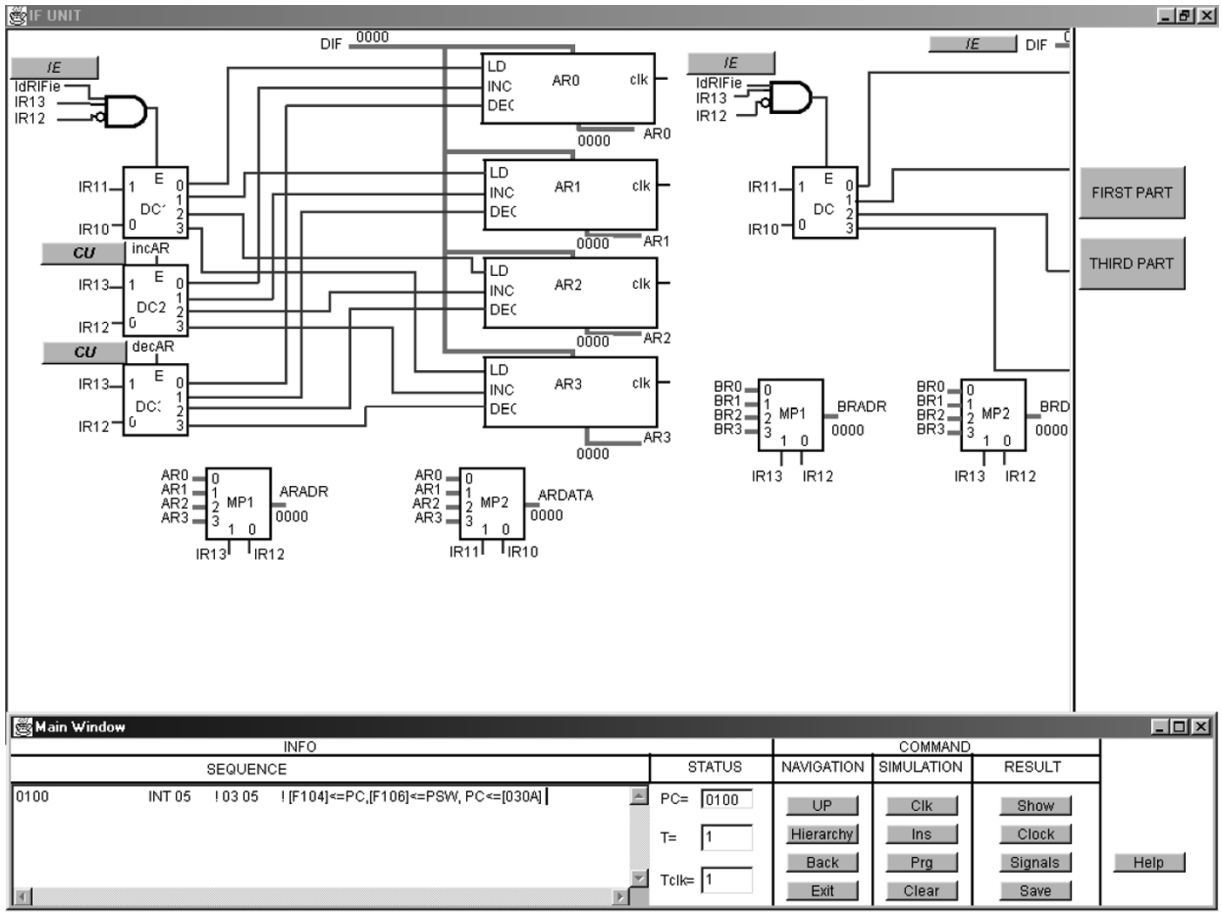

Fig. 4. Part of the processing unit of the processor instruction fetch unit.

is the processor module for which the user goes from the first to the second level through an intermediate level screen, which gives the structure of the processor module at the level of units (Fig. 5). For each signal coming from another part of the computer system, a link button with the name of the module where the signal is generated is provided (Fig. 4); this scheme provides an easy and fast navigation through screens. A single line changes its color depending on the current value (logic "0" or "1"), and a bus is accompanied by its current value.

The main window in the lower part of the screen (Fig. 4) shows the status of simulation (PC—program counter, T—step counter, Tclk—processor clock cycles executed), the control signals generated for that clock period, and a brief explanation of the actions to take place during that clock period in the

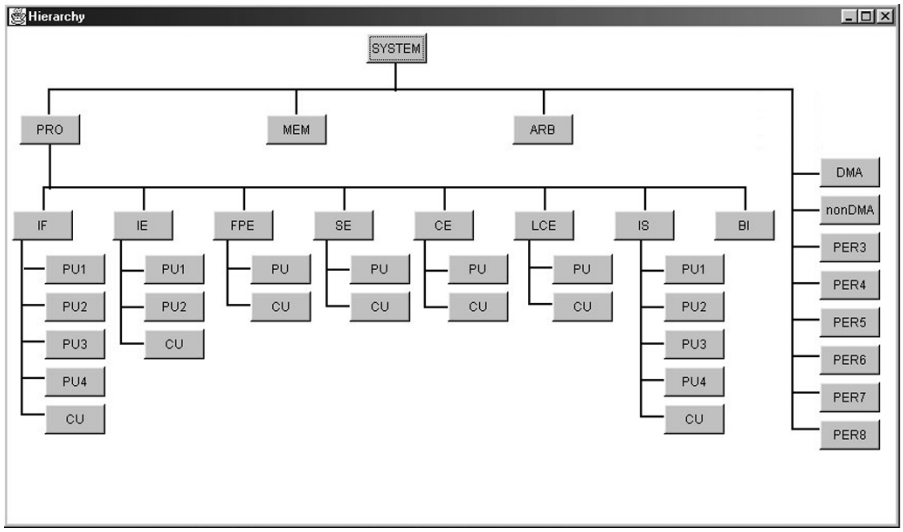

Fig. 5. Button hierarchy activated.

```
0100   LOADB imm(10)          ! 20 D0 10     !acc<=10h
0103   MULU i16(XR0,003C)     ! 34 83 3C 00  !acc<=acc*mem[XR0+003C]
0107   STOREB b16(BR3,00A0)   ! 22 6C A0 00  !mem[BR3+00A0]<=acc
```

Fig. 6. Program illustrating operations and addressing modes.

sequence box. The command buttons support navigation (UP, HIERARCHY, and BACK), stop of simulation (EXIT), simulation (CLK, INS, PRG, and CLEAR), and examination of simulation results (SHOW, CLOCK, and SIGNALS), saving the simulation context (SAVE). More details regarding simulator capabilities can be found in [17].

V. Laboratory Organization

Each exercise has four components: prelaboratory preparation, in-laboratory knowledge assessment, in-laboratory assignment, and written report. To prepare for a particular laboratory, the students must review related material from lectures and the textbook and read the related sections from the laboratory manual. They can also access the EDCOMP simulator from home and use it for self-study. Each laboratory assignment is preceded by a short computer-based test designed to verify whether the students understand the topic covered in the assignment [18]. After passing the test, the students select a predefined experiment, which results in the initialization of the computer system from a file. Then, they execute the programs, either at the instruction level or at the clock level. Before, during, and after the simulation, the students are requested to examine the values of relevant memory locations and registers in the processor and the I/O units, follow the values of selected signals at combinational and sequential circuits, and draw their timing diagrams. In some of the experiments, the students are also asked to develop their own assembly programs using the editor, the translator, and the loader and to verify the programs' correctness and determine performance. Based on the observations made during the experiment, they answer questions relevant to the topic and turn in a written report. The following section gives examples of typical laboratory experiments.

VI. Using Educational Computer Simulator: Examples

To illustrate the use of the simulator in learning computer architecture and organization, this section gives several examples both of the computer architecture and computer organization experiments.

A. The Computer Architecture Experiments

The programs in this group illustrate the programmer's view of the processor, the interrupt, and the I/O. The simulation is run instruction by instruction.

1) The Processor: These programs include various instructions, addressing modes, data types, instruction formats, and programmable registers. There are two groups of programs. The first group includes instructions with simple algorithms of operation, such as transfer, arithmetic, logic, shift, rotate, and unconditional and conditional jump instructions. These programs utilize all addressing modes given in Section III and operate on integers. The second group includes instructions with more complicated algorithms of operations; simple addressing modes; and integer, floating-point, and string data types. The instructions used are multiplication and division with unsigned and signed integers, the push/pop from the stack, the jump/return from subroutine, the arithmetic operations with the floating-point numbers and the conversions between integers and floating-point numbers, and finally, string and loop control instructions.

Fig. 6 shows a program illustrating execution of load byte (**LOADB**), multiply unsigned integers (**MULU**), and the store byte (**STOREB**) instructions; the effects of the immediate (**imm(10)**), the index with 16-b displacement (**i16(XR0,003C)**), and the base with 16-b displacement addressing modes (**i16(BR3,00A0)**); and the use of the index (**XR0**) and the base (**BR3**) programmable registers. The **XR0** and **BR3** registers are preloaded with values 0001 h and

```
0100     INT 05   ! 03 05    ![F104]<=PC,[F106]<=PSW,PC<=[030A]
0102       ...
           ...
0200     RTI      ! 04       !PSW<=[F106],PC<=[F104]
0201       ...
```

Fig. 7. Program illustrating the interrupt instruction.

```
0100       LOADW imm(0FFF)  ! 21 D0 FF 0F    ! acc <= 0FFF
0104       STOREW abs(AR0)  ! 23 14     ! AR0 <= acc
0106       LOADB imm(03)    ! 20 D0 03 ! acc <= 03
0109       STOREB abs(DR0)  ! 22 10     ! DR0 <= acc
010B       LOADB imm(04)    ! 20 D0 04 ! acc <= 04
010E       STOREB mem(F100) ! 22 30 00 F1! [F100] <= acc
0112 loop: LOADB imm(01)    ! 20 D0 01 ! acc <= 01; mask for Ready bit
0115       AND mem(F102)    ! 40 30 02 F1! acc <= acc AND [F102]
0119       BEQL loop        ! 02 F9     ! if not zero goto loop
011B       LOADB pri(AR0)   ! 20 C0     ! AR0 <= AR0 + 1, acc <= [AR0]
011D       STOREB mem(F104) ! 22 30 04 F1! [F104] <= acc; send byte
0121       LOADB abs(DR0)   ! 20 10     ! acc <= DR0
0123       DEC              ! 3B        ! acc <= acc - 1
0124       STOREB abs(DR0)  ! 22 10     ! DR0 <= acc
0126       BNEQ loop        ! 03 E9     ! if not zero goto loop
0128       LOADB imm(00)    ! 20 D0 00 ! acc <= 00
012B       STOREB mem(F100) ! 22 30 00 F1! [F100] <= acc
```

Fig. 8. Program illustrating the I/O.

00A8 h, respectively. As a result, the index addressing mode gives the effective memory address 003D h, from which data 02 h is read, and the base addressing mode address 00A8 h at which the result 20 h is stored.

2) The Interrupt: These programs demonstrate topics such as the jump to an interrupt routine and return from the interrupt, for various types of internal and external interrupts, the selective and complete masking of interrupt requests from the I/O units, the servicing and the nesting of multiple interrupt requests, and the execution of the interrupt instruction. They also demonstrate how the starting address of an interrupt routine can be found by using the interrupt vector table or by polling the I/O units or by a combination of both. Fig. 7 shows a program that illustrates the effects of execution of the interrupt instruction (**INT 05**). This instruction saves the contents of the progam counter (PC) and program status word (PSW) registers on the stack and causes a jump to the interrupt routine at the address 0200 h, loaded from entry 5 of the interrupt vector table. The preloaded values of the stack and the interrupt vector table pointers are F103 h and 0300 h, respectively. Since each address is 2 B long, entry 5 is at addresses 030A h and 030B h. The interrupt routine contains only the return from interrupt instruction (**RTI**), which restores the contents of the PC and PSW registers from the stack and causes a return to the instruction at address 0102 h.

3) The Program Controlled Input/Output: These programs demonstrate how blocks of data are transferred within the computer system using the non-DMA and DMA controllers. Transfers with the non-DMA controller are carried out between the peripheral device of an I/O unit and the memory, using both the polling and the interrupt technique. Transfers with the DMA controller are carried out between the peripheral device and the memory and between two parts of memory with both the cycle stealing and the burst modes of operation.

Fig. 8 shows a program where polling is used to transfer a block of 3 B starting at memory address 0FFF h to the peripheral device of an output unit using the non-DMA controller. The first part is the initialization of the processor's **AR0** and **DR0** registers and the controller's control register at address F100 h with the block address, the block size, and the start mode of operation. The second part is the inner loop where the polling of the controller's status register at address F102 h is performed. The third part is the outer loop where a byte is transferred into the controller's data register at address F104 h, register **AR0** is incremented, and **DR0** is decremented. The fourth part is the loading of the controller's control register at address F102 h with the stop mode of operation.

B. The Computer Organization Experiments

The programs in this group are created to illustrate the phases of the instruction execution and the bus arbitration and cycles. They are executed at the clock level.

1) The Phases of Instruction Execution: The instruction and operand fetch phase and the operation execution phase are illustrated by executing the programs explained in Section VI-A-1) and the interrupt handling phase by executing the programs explained in Section VI-A-2). During execution of these phases, the students follow the appropriate sequence of control signals (Fig. 9).

The program in Fig. 6 illustrates the realization of the LOADB, STOREB, and MULU instructions and immediate, base, and index addressing with 16-b displacement. In the IF unit, the students follow how the first, second, third, and fourth byte of an instruction are read; the operand address is formed depending on the addressing mode specified; the operand is read; the IE unit is activated; and the IF unit deactivated. In the IE unit the students follow how the operation is executed; the IS unit is activated; and the IE unit is deactivated. In the IS unit, they find that there is no interrupt generated; therefore, the IF unit is activated, and the IS unit deactivated.

The program in Fig. 7 illustrates the realization of the interrupt handling phase and the operation of the IS unit. The INT instruction goes through the IF and IE units as described in the previous paragraph. However, when it arrives in the IS unit, the

```
        step0   br(if OEFIF then step0) ! IF waits to be activated !
! Read 1, 2, 3 or 4 bytes of instruction and load into instruction registers IR0, IR1, IR2 and IR3 !
! Read 1. byte and load into IR0 !
        step1   IdMARif, incPCIF        ! MAR<=PCIF, PCIF<=PCIF+1 !
        step2   readif                  ! MDR<=MEM !
        step3   IdIR0                   ! IR0<= MDR !
        step4   IdIRRCODIF              ! Set IRRCODIF if operation code error !
        step5   br(if IRRCODIF then stepFF)  ! Finish activities in IF if operation code error !
        step6   br(if INS1B then stepFF)     ! Finish activities in IF if instruction length is 1 byte !
!
! Read 2., 3. and 4. byte and load into registers IR1, IR2 and IR3 !
        step7   IdMARif, incPCIF        ! MAR<=PCIF, PCIF<=PCIF+1 !
        ...
! Read operand depending on addressing mode specified by active value of one of signals
short,..., base,..., index,... etc. !
        step14  case (..., short, ..., base, ..., index, ...)
                (short, step18)...(base, step64)...(index, step84)...
        ...
```

Fig. 9. Sequence of control signals for IF.

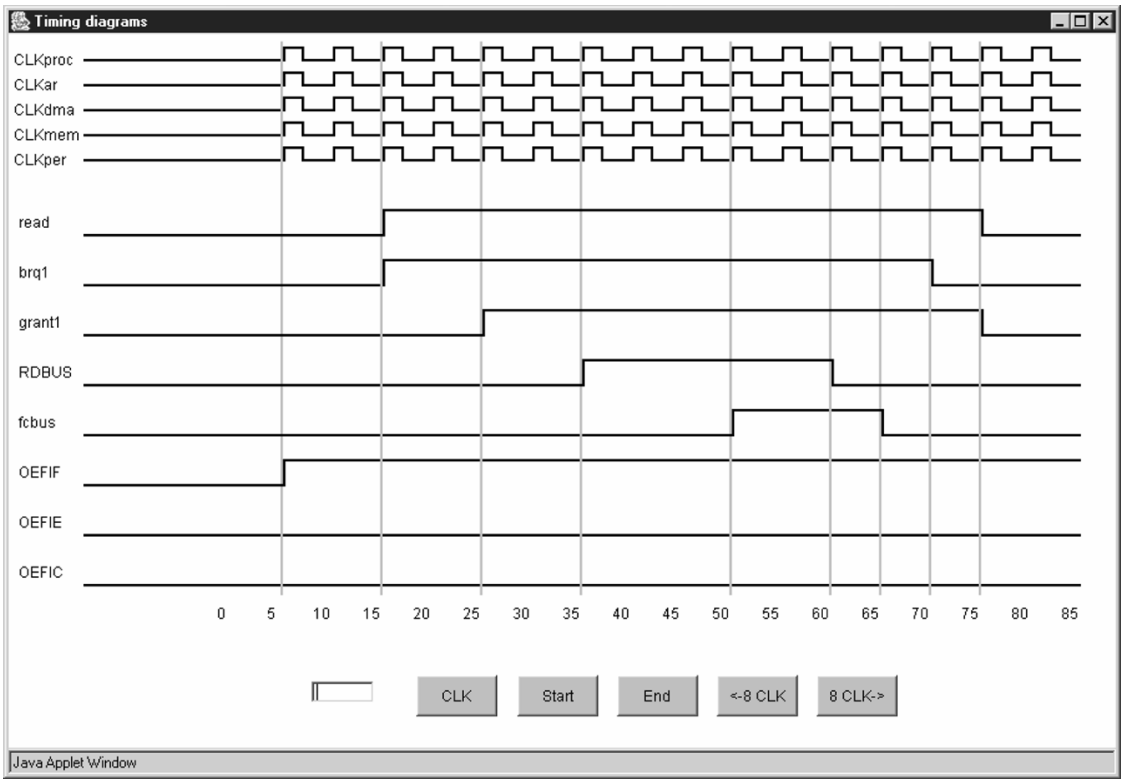

Fig. 10. Timing diagrams during the read bus cycle.

presence of the interrupt is found. Therefore, the PC and PSW registers are saved on the stack; value 0200 h is loaded in the PC register from entry 5 in the interrupt vector table; the IF unit is activated; and the IS unit is deactivated. The next instruction executed is the RTI instruction. The instruction is first read in the IF unit; then, it is executed in the IE unit by restoring the contents of the PC and PSW registers from the stack; and finally, since it is found in the IS unit that there is no interrupt, the IF unit is activated, and the IS unit is deactivated.

2) The Bus Arbitration and Bus Cycles: The bus arbitration is demonstrated with two experiments, where the dummy controllers appear as the bus masters. In the first experiment, a single bus master participates in the arbitration. This experiment demonstrates how the bus master sends the bus request; the arbiter acknowledges with the bus grant; the bus master keeps the bus busy during the bus cycle; the bus master abolishes the bus request; and the arbiter does the same with the bus grant. In the second experiment, two bus masters participate in the arbitration. The exchange of signals between the bus masters and the arbiter is very similar to the one in the previous experiment. The difference is that one bus master has to wait to obtain the bus grant until the other one completes its cycle on the bus.

The bus cycles demonstrated in the experiment are the read and write cycles between the processor as the bus master and the memory as the bus slave, and the interrupt vector number acquisition cycle between the processor as the bus master and the I/O unit as the bus slave. The experiment demonstrates how the address, data, and control signals are exchanged between a bus master and a bus slave for each of the cycles. Fig. 10 shows the timing diagrams during the read bus cycle.

VII. EDCOMP Assessment

Qualitative and quantitative evaluations of the proposed educational approach have been conducted. The qualitative evaluation included a number of student surveys and discussions with fellow instructors who teach courses that directly or indirectly have this course as a prerequisite. The surveys were designed to learn what students perceived as a good educational tool and how they assessed the overall effectiveness of this approach. The students were also asked to specify things they liked the most and the least. These forms have been a valuable source for the authors, and some of students' suggestions have been implemented in later versions of the EDCOMP. The majority of students found the graphical representation very instructive and the user interface easy to use. The fellow instructors reported that students who used EDCOMP were better prepared and had deeper understanding of basic concepts so that the time needed for revision could be reduced. The School of Electrical Engineering, University of Belgrade, conducts a survey at the end of each school year where the students assess the quality of all laboratory courses they have taken. Each laboratory is graded using a five-point scale (excellent, very good, good, fair, and poor). The authors compared the students' ratings taken in the period of 2001–2003, when EDCOMP was used, versus ratings collected during a four-year period before EDCOMP was introduced. The average grade for the laboratory has increased from 4.1 to 4.7. However, the significantly increased number of students who rated the laboratory as excellent and very good (70%) is partially offset by an increase in the number of students who rated the laboratory as poor (17%). The latter found that EDCOMP required a significant effort on the students' side. Overall, the students ranked this laboratory as the second most useful in the second year—up from sixth.

For the quantitative evaluation, the authors compared the average total score in the course for the two periods described previously. The number of students taking this course is about 450 per year. The average total score on a scale 0 to 100 has increased from 78 to 86 after the introduction of EDCOMP. In addition, the number of students who successfully passed the exam has increased from 72% to 85%.

VIII. Conclusion

This paper introduced a flexible, Web-based, educational environment designed to help teaching and learning in computer architecture and organization courses. The environment goes beyond existing simulators by covering a broad range of relevant topics and providing visualization of the computer system functioning with various levels of details. This approach allows students to associate knowledge gained in class with practical examples and promotes active learning: to pass laboratory assignments successfully, students must become engaged learners and assimilate related knowledge before coming to the laboratory. This approach also provides a vertical integration with prerequisite courses in digital logic design and multilevel comprehension of the computer system, from the system level down to the gate level. Students can start from the programmer's view of the computer system, write programs, and follow the program execution at the highest level. If they want to examine the implementation details, they navigate through the system hierarchy down to the digital logic level. The sophisticated process initialization helps instructors in preparing laboratory experiments. Although EDCOMP is developed to satisfy specific needs of the School of Electrical Engineering at the University of Belgrade, its built-in flexibility allows its use in other introductory courses in computer architecture and organization.

References

[1] A. Clements, "The undergraduate curriculum in computer architecture," *IEEE Micro*, vol. 20, no. 3, pp. 13–22, May/Jun. 2000.

[2] (2004, Jun.) Computing curricula—Computer engineering, IEEE Computer Society and ACM. [Online]. Available: http://www.eng.auburn.edu/ece/CCCE

[3] D. Burger and T. Austin, "The SimpleScalar Tool Set Version 2.0," Univ. of Wisconsin, Madison, Tech. Rep. CS-TR-97-1342, 1997.

[4] M. Rosenblum, E. Bugnion, S. Devine, and S. A. Herrod, "Using the SimOS machine simulator to study complex computer systems," *ACM Trans. Modeling Computer Simulation*, vol. 7, no. 1, pp. 78–103.

[5] Y. Zhang and G. B. Adams, "An interactive, visual simulator for the DLX pipeline," *IEEE TCCA Newslett.*, pp. 9–12, Sep. 1997.

[6] R. N. Ibbett, "Hase DLX simulator," *IEEE Micro*, vol. 20, no. 3, pp. 57–65, May/Jun. 2000.

[7] J. L. Bechennec, "ASF: A teaching and research object-oriented simulation tool for computer architecture design and performance evolution," in *Proc. Workshop Computer Architecture Education (WCAE-98)*, Barcelona, Spain, Jun. 27, 1998, pp. 93–96.

[8] P. Verplaetse, J. V. Campenhout, and H. Neefs, "ESCAPE: Environment for the simulation of computer architecture for the purpose of education," in *Proc. Workshop Computer Architecture Education (WCAE-98)*, Barcelona, Spain, Jun. 27, 1998, pp. 42–47.

[9] E. Pastor, F. Sanchez, and A. M. d. Corral, "A rudimentary machine. Experiences in the design of a pedagogic computer," in *Proc. Workshop Computer Architecture Education (WCAE-98)*, Barcelona, Spain, Jun. 27, 1998, pp. 31–36.

[10] J. R. Arias and D. F. Garcia, "Introduction computer architecture education in the first course of computer science career," in *Proc. Workshop Computer Architecture Education (WCAE-98)*, Barcelona, Spain, Jun. 27, 1998, pp. 82–87.

[11] C. Yehezkel, W. Yurcik, and M. Pearson, "Teaching computer architecture with a computer-aided learning environment: State-of-the-art simulators," presented at the Int. Conf. Simulation Multimedia in Engineering Education (ICSEE), Phoenix, AZ, Jan. 2001.

[12] D. Ellard, D. Holland, N. Murphy, and M. Seltzer, "On the design of a new CPU architecture for pedagogical purposes," in *Proc. 9th Workshop Computer Architecture Education*, Anchorage, AK, May 2002, pp. 27–33.

[13] R. Hodson and J. Hereford, "Interactive CPU simulator for computer organization instruction," presented at the 3rd Workshop Computer Architecture Education (WCAE), San Antonio, TX, Feb. 1997.

[14] G. Adams and H. Dietz. (1997) Compiler/Architecture Simulation for Learning and Experimenting. [Online]. Available: http://shay.ecn.purdue.edu/~casle/

[15] C. J. Hughes, V. S. Pai, P. Ranganathan, and S. V. Adve, "Rsim: Simulating shared-memory multiprocessors with ILP processors," *IEEE Computer*, vol. 35, no. 2, pp. 40–49, Feb. 2002.

[16] J. Djordjevic, A. Milenkovic, and N. Grbanovic, "An integrated environment for teaching computer architecture," *IEEE Micro*, vol. 20, no. 3, pp. 66–74, May/Jun. 2000.

[17] J. Djordjevic, B. Nikolic, and A. Milenkovic. (2004) Educational system for teaching computer architecture and organization—Demo version. [Online]. Available: http://electra.etf.bg.ac.yu:8080/SIMCISCO/SIMCISCO.html

[18] J. Djordjevic, A. Milenkovic, I. Todorovic, and D. Marinov, "CALKAS: A computer architecture learning and knowledge assessment system," *IEEE TCCA Newslett.*, pp. 26–29, Jul. 2000.

Jovan Djordjevic received the B.Sc. degree in electrical engineering from the University of Belgrade, Belgrade, Serbia and Montenegro, and the M.S. and Ph.D. degrees in computer science from the University of Manchester, Manchester, U.K.

He is currently an Associate Professor of Computer Engineering at the School of Electrical Engineering, University of Belgrade. His research interests include computer architecture, parallel computer systems, digital systems simulation, and distance learning.

Bosko Nikolic received the B.Sc. and M.S. degrees in electrical engineering from the University of Belgrade, Belgrade, Serbia and Montenegro.

He is currently a Research Assistant in the Department of Computer Engineering at the School of Electrical Engineering, University of Belgrade. His research interests include computer architecture, digital systems simulation, and the programming language Java.

Aleksandar Milenkovic (M'96) received the B.Sc., M.S., and Ph.D. degrees in computer engineering from the University of Belgrade, Belgrade, Serbia, in 1994, 1997, and 1999, respectively.

He previously held faculty positions with the University of Belgrade and Dublin City University, Dublin, Ireland. He is currently an Assistant Professor of Electrical and Computer Engineering at the University of Alabama, Huntsville. His research interests include computer architecture, performance analysis, embedded systems, very large scale integration (VLSI), and parallel and distributed systems.

Dr. Milenkovic is a Member of the Association for Computing Machinery (ACM), the IEEE Computer Society, and Eta Kappa Nu.