An Integrated Educational Environment for Teaching Computer Architecture and Organisation

Jovan Djordjevic, Aleksandar Milenkovic, Nenad Grbanovic
Faculty of Electrical Engineering, University of Belgrade

Abstract: *The paper presents an integrated educational environment for teaching courses in computer architecture and organisation. The integrated educational environment includes the Integrated Educational Computer System (IECS), the reference manuals for it, the Software Package of the Integrated Educational Computer System (SPIECS), a set of laboratory experiments, and the Computer Architecture Learning and Knowledge Assessment System (CALKAS). The IECS covers the basic structure of a computer system: processor, memory, input/output subsystem, and bus. The accompanying manuals provide all implementation details with the appropriate circuits drawings and detailed descriptions. The SPIECS includes the graphical simulators of the IECS down to the register transfer level and the tools needed to select, configure, initialise and run the simulation of the appropriate parts of the IECS. Various laboratory experiments are prepared for students' practical work with the IECS. The CALKAS program keeps the complete evidence about each student's work in the laboratory, assesses its knowledge at the end of each laboratory experiment and produces various reports at the completion of all laboratory experiments.*

1. Introduction

Two courses in Computer architecture and organisation have been lectured by the authors at the Faculty of Electrical Engineering, University of Belgrade. The first course in Computer architecture and organisation is a second year undergraduate course attended by the students at the departments of Communications, Automation, Electronics, and Computer Science. The students of these departments acquire the knowledge necessary for following such a course through the first year course in Fundamentals of Computer Science and the second year course in Programming methodology and languages. The course in Computer architecture and organisation covers the basic concepts of the most commonly found structure of a computer, which includes the processor, the memory, the input/output subsystem, and the bus [1]. The second course in Computer architecture and organisation is attended by the students at the Department of Computer Science and Engineering. This course goes one step further covering topics such as the architecture and organisation of CISC and RISC processors, the organisation of pipelined processors, the storage system, the interconnection networks, and the memory system [2].

Generally, a great problem in teaching any course in the field of computer architecture and organisation is how to provide means which would facilitate the students to make a cognitive leap from the blackboard description of the architecture and organisation of a computer to its utilisation as a programmable device and connect their theoretical knowledge with practical experience. This problem has been tackled by a number of people involved in computer architecture and organisation education resulting in various educational tools that simulate computer systems. They are very different concerning the scope, the complexity of target architecture, the simulation level, the user interface, the software and hardware requirements, the required knowledge, etc.

One group of the educational tools is based on rudimentary architectures with less than 20 instructions supported [3, 4]. They support simulation at the functional level, and allow one to follow the processor dataflow in a graphic manner. The educational systems presented in [5, 6] are based on the DLX instruction set [2]. The system presented in [5] supports the microcode or pipelined

organisations. For each organisation there is a window showing the system structure, register states and signal lines connecting system elements. Viewing and editing of the memory are also possible, as well as viewing of the pipeline activity diagram. The simulator presented in [6] provides animated versions of the key figures and tables from [2], and allows a user to follow details of the pipeline activities during the program execution at the clock or instruction level.

Several very powerful simulators that demonstrate the functioning of virtual or commercial processors such as SparcV8 [7] and PowerPC601 [8] have been developed. These simulators provide rich instruction sets, numerous addressing modes and data types, and high simulation speeds. In addition, these simulators are capable of running Unix programs, C compilers, and SPEC92 benchmark set. The primary goals of these simulators are as fast as possible simulations of the target systems themselves and statistical analysis and time diagram viewing, while the visualisation is either non-existent or at best rudimentary.

The HASE system [9], based on an object-oriented system for database management which runs in X-Windows/Motif environment, is aimed at high level simulation and visualisation of the simulated project. A user has access to the read-only HASE component library, as well as the possibility of creating user component libraries. The system architecture that has to be simulated is created using a graphical program in which system components are selected, placed, and linked. After defining the system configuration, a set of files that describes the system is created, and simulation can be performed. During the simulation, an event trace file is created containing all required data about the current simulation run. This file may be used as an input for the program that animates the working of the simulated system, or the program for viewing time diagrams. Numerous statistical analyses are also available. A system with a similar approach is presented in [10]. A C++ application is developed that enables a user to create system blocks, configure the system and run simulations. Several classes are available: block class, link class, data transfer class, data class and simulator class. Using available classes, the user creates blocks, links between blocks, defines rules of data transfer between blocks, data types, and finally configures the whole system. Simulation results are shown as textual reports with values of parameters relevant to the objects used.

The above-described systems provide various means for simulating digital systems. They usually offer some kind of tools for creating hardware component libraries, viewing simulation results, and carrying out statistical analysis of system performance. They range from those aimed at a sophisticated simulation and performance analysis of complex systems, to those devised for simpler systems easily understood by users. However, the authors strongly feel that a computer architecture and organisation environment should provide support for achieving the following objectives. First, it has to cover the computer architecture and organisation topics lectured in the above-mentioned courses and summarised in Table 1. Second, it should have a graphical presentation of a computer system, from the block to the register transfer level. Third, it must provide means to follow the functioning of the system at the program, instruction and clock cycle levels. An analysis of the above described systems had shown that none of them meets the objectives stated, which led to the development of an integrated educational environment for teaching computer architecture and organisation.

The integrated educational environment includes the Integrated Educational Computer System (IECS), the reference manuals for it, the Software Package of the IECS (SPIECS), a set of laboratory experiments, and the Computer Architecture Learning and Knowledge Assessment System (CALKAS). The IECS, described in Section 2, covers all topics of interest and its detailed description is given in the accompanying manuals. The SPIECS, given in Section 3, includes the tools needed to select, configure, and initialise the appropriate parts of the IECS, and the graphical simulators of the IECS. A set of laboratory experiments and the CALKAS program, presented in Section 4, facilitate students' practical work with the IECS. Section 5 contains the conclusion.

Table 1. Computer architecture and organisation topics.

Processor		Memory	Bus	Input/Output
Architecture	**Organisation**			
•CISC vs. RISC •Programming registers •Data types •Instruction formats •Addressing modes •Instruction sets •Interrupt mechanisms	•Processing unit (PU): pipelined vs. nonpipelined •Control unit for pipelined PU •Control unit for nonpipelined PU: hardwiring, microprogramming (horizontal, mixed, vertical formats), nanoprogramming	•Memory management and TLB units •Cache memory •Memory interleaving	•Asynchronous vs. synchronous •Atomic vs. split-transaction •Arbitration	•Programmed I/O: polling vs. interrupt driven •Block transfers with DMA •Peripheral device and DMA controllers implementations

2. The Integrated Educational Computer System

The integrated educational computer system (IECS) is made up of three self-contained systems:
- the RISC processor based educational computer system (the RCS system),
- the CISC processor based educational computer system (the CCS system), and
- the educational hierarchical memory system (the HMS system).

The basic features of each of these systems are given in the following.

The RCS system and the CCS system have similar structures and are made up of the processor, the memory and the input/output subsystem while the communication between them is carried out through the asynchronous bus [11]. The major difference between them is in the processors architecture and organisation.

The RCS system processor architecture is the load/store type one with general-purpose registers and the usual registers, such as the program counter (PC), the stack pointer (SP), the program status word (PSW), etc. Data types supported are 16-bit signed and unsigned integers. The instruction format is the three address one and the instruction length is 32 bits. The addressing modes achieved for the Load and Store instructions are the immediate, memory direct, register indirect, and register indirect with displacement, while all the remaining instructions implicitly use the register direct addressing mode. The instruction set includes the transfer, arithmetic, logic, shift, rotate and control instructions. There are internal and external interrupts. Two types of processor organisation have been implemented, the nonpipelined one and the pipelined one. The nonpipelined processor organisation is the one with the processing and control units. The processing unit is made up of the register file (RF), the execution unit (EXU), the interrupt service unit (ISU), and the bus interface unit (BIU) interconnected with the 16-bit internal bus. For the processing unit four types of control units are designed, one of them uses the hardwired technique, while the remaining three the microprogramming technique with the mixed and vertical formats of microinstructions, and nanoprogramming. The pipelined processor organisation is made up of five stages. Each stage corresponds to a separate phase of an instruction execution. These processor stages and instruction phases are the instruction fetch, the instruction decode and operands read, the operation execution, the memory access, and the result write. In order to avoid structural hazards, separate instruction and data caches are implemented, and the register file is implemented to allow two registers to be read and one to be written at the same time. The read after write data hazard is eliminated by using the data forwarding technique. The control hazards are minimised by using the dynamic prediction mechanism with the branch target buffer.

The CCS system processor architecture is the one with separate data, address, base and index registers, and the usual registers such as the program counter, the stack pointer, the program status word, etc. Data types supported are 8-bit signed and unsigned integers, 16-bit floating point numbers and strings of characters. The length of the instructions is 1, 2, 3 and 4 bytes. The instruction format is the one-address one. The addressing modes are the register direct, register indirect, memory direct, memory indirect, base, index, base index, 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. There are internal and external interrupts. The processor organisation is the one with separate processing and control units. The processing unit is made up of the instruction fetch, decode, operand address calculation and operand reading unit, five execution units (Integer, Floating-point, String, Control, and Loop Control), the interrupt service, and the bus interface unit. Each of these units has its own control unit, being realised by using either the hardwired technique or some of the various types of microprogramming techniques.

The structures of the memory, the input/output subsystem and the bus are very similar for the RCS and CCS systems. The capacities of the memories are 128KB and 64KB, respectively. The input/output subsystem supports 12 peripheral devices. Eight of them are with peripheral device controllers and four with direct memory access controllers. The components are interconnected through an asynchronous bus.

The HMS system is made up of the virtual memory and translation lookaside buffer, the cache memory and the main memory. The initial idea was to develop such a hierarchical memory system where the process of accessing the translation lookaside buffer, the cache memory and the interleaved memory is realised as a whole. Although the realisation of such an HMS environment was feasible, the authors felt that such a system would be too complex and very difficult to be used by the students. Therefore, it was decided to split up the HMS environment into three separate entities for the virtual memory and translation lookaside buffer, the cache memory and the interleaved memory.

The virtual memory and translation lookaside buffer (TLB) are demonstrated by using the paged, segmented and segment-paged organisations and the TLBs with the direct, associative, and set-associative mappings. For each of them, the complete procedure of the virtual to real address translation has been carried out, according to the characteristics of the TLB and virtual memory. It most frequently includes the reception of the virtual address from the processor, the accessing of the appropriate descriptor from the TLB, the access rights and bounds checks, and the real address generation. Occasionally, the descriptor is accessed from the segment and/or page table and the TLB updated. The TLB entry to be replaced with a new descriptor according to the LRU and FIFO algorithms where appropriate.

The cache memory is demonstrated by using the direct, associative and set associative mappings. The complete procedure of receiving a main memory request from the processor, checking the cache memory for a hit, and performing the read or write is carried out according to the specific features of each mapping technique. The FIFO replacement algorithm is implemented where needed and both the write-back and write-through main memory updating techniques used. The block transfer form the main memory in the case of a miss is realised, including the block buffering, critical word first, and early processor start techniques.

The interleaved main memory is demonstrated in a system made up of 16 units, 16 memory modules, the bus and the arbiter. Each unit can be configured to generate either the single word accesses or the block accesses and includes the bus interface and the requester. The bus interface contains all circuitry necessary to perform the appropriate operations when the unit is either a master or a slave. The requester simulates parts of a processor or a direct memory access controller, which

generate the memory read and write requests and accepts data returned as a response to a read request. A memory module is made up of the bus interface, which contains all circuitry necessary to perform the appropriate operations when the unit is either a slave or a master, and the RAM memory. Five possible ways of interleaving memory modules can be specified. The bus is realised as a split transactions synchronous bus. The arbiter realises the parallel arbitration between all units and modules.

3. The Software Package of the Integrated Educational Computer System

The Software Package of the Integrated Educational Computer System (SPIECS) includes the graphical simulators of the appropriate parts of the IECS and the tools needed to select, set up and run the appropriate simulator. The work with the SPIECS must be performed in three steps. The first step is used to select the simulator of one of three educational computer systems. In the case of the HMS system one must further select one of three simulators for the virtual memory and TLB, the cache memory, and the interleaved memory. For the virtual memory and the cache memory, further selection is needed for one of three types of TLBs and cache memories, respectively. The second and third steps are provided to set up and run the simulator, respectively. Their basic features are given in the following. The features and their use are very similar for all three systems. Therefore, in order to keep the presentation clear and consistent whenever illustrations are given the RCS system is used.

3.1 The simulator set up

The simulator set up includes the steps to configure and initialise the appropriate simulator of the IECS.

The first step of the simulator set up is to configure the selected simulator if required. It allows one to choose the number of peripheral device and DMA controllers, to select the non-pipelined or pipelined implementation of the processing unit, and to select one of four implemented control units to be used in the system with nonpipelined processing unit (Figure 1), and to define clock and access times (Figure 2).

The second step of the simulator set up is to initialise the selected simulator. It consists of the initialisation of the processor, the memory and the peripheral device and DMA controllers. The initialisation of the processor involves the loading of the program counter, interrupt vector table pointer, interrupt mask register, stack pointer, general purpose registers etc. The initialisation of the memory includes the loading of the appropriate memory locations with the binary values of programs, the data to be used during the execution of a program, the addresses of interrupt routines in the interrupt vector table, the data to be sent to the output units etc, and can be done either manually or using the Editor, Assembler, Linker, and Loader. The initialisation of the peripheral device and DMA controllers includes the loading of the simulated input peripheral device with the sequence of data that this device will generate and the time when the data should be generated.

The simulator set up can be carried out either partially or completely. If the partial simulator set up is chosen one has to do it separately for each part of the RCS system such as processor registers, memory locations, peripheral device and DMA controllers. Otherwise, it is done completely for all parts of the system at once. The partial and complete set ups can be carried out either interactively or from files. The simulator set up features are suited to meet the needs of both instructors and students. The primary aim is that instructors extensively use them in preparing laboratory experiments. As for the students, they should first get acquainted with SPIECS, by executing laboratory experiments prepared by instructors, and later, as part of solving tasks assigned to them, they can, among other things, carry out the complete simulator set up.

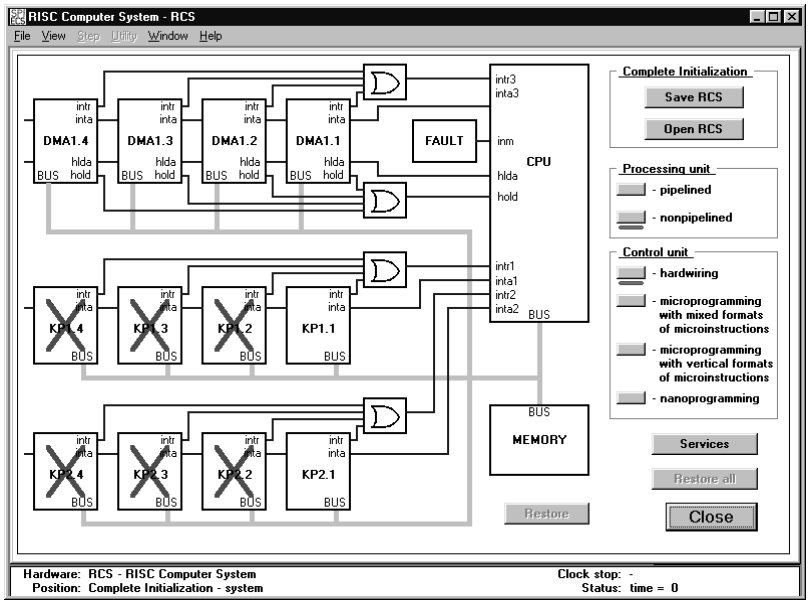

Figure 1. The RCS system configuration.

This screen is used to configure the I/O subsystem, the processing and control units. The I/O subsystem can contain up to four DMA controllers, denoted as DMA1.x and eight peripheral device controllers, denoted as KP1.x and KP2.x, where x=1,..,4. Each of them can be disconnected or reconnected into the system by consecutive mouse clicks on the appropriate box. The Processing unit box allows one to choose the pipelined or non-pipelined implementation of the processing unit. If the nonpipelined implementation is specified, one has to select from the Control unit box which of four implemented control units is to be used with the processing unit. The Complete Initialisation box includes buttons Open RCS and Save RCS allow one to configure and initialise the whole system from a file, or to save the current state into a file. Buttons Restore and Restore All facilitate to cancel some or all interactively made changes, respectively. Button Services allows one to move to the clock and access times screen (Figure 2).

3.2 The simulator run

The simulator allows one to follow the values of signals of the RCS system at various levels during the execution of a program after a clock, an instruction or a complete program. Because a limited number of elements can be displayed on a screen, a hierarchical scheme of the RCS is developed. Figure 3 shows the root of the hierarchical scheme of the RCS with all relevant units. Each screen, in general, is made up of the title bar, menu, and five windows.

The largest window in the upper left part of the screen, named the Block diagram window, contains either only a composition of combinatorial and sequential circuits, if this is a leaf block in the hierarchical scheme, or a composition of sub blocks, that can be further selected, and combinatorial and sequential circuits, if this is not a leaf block in the hierarchical scheme.

The upper right window, named the Hierarchy window, shows the hierarchical scheme of the CPU. Each block from the hierarchical scheme is further presented with one or more screens. The traversing through the blocks of the hierarchical scheme can be achieved by selecting the appropriate block, at any moment. The name of the block shown in the Block diagram window is given in red.

The lower left window, named the Info and Command window, is divided into the Info window at its left hand side and the Command window at its right hand side. The Info window contains the Sequence window and the Status buttons. The Sequence window gives the value either of the microprogram counter or step counter, then the control signals generated for that clock period and, finally, a brief explanation of the actions that are going to take place during that clock period. The Status buttons PC, T, and Tclk display the current values of program counter, the step counter and the

number of the CPU clock periods executed, respectively. The Command window contains three groups of command buttons: Navigation, Simulation and Miscellaneous. The Navigation command button UP allows one to move from the current screen to the screen one level up in the hierarchy, while button Main brings back the root hierarchical screen (Figure 3). The Simulation command buttons Clk+, Ins+ and Prg+ allow one to continue with the simulation just till the first clock to appear or with the number of clock periods required to complete the current instruction or the complete program, respectively. The Miscellaneous command buttons are More, Clear and Help. Button More opens the window that allows one to select one of three screens. They further facilitate to show and set the values of memory locations, processor registers, and draw the timing diagrams of a selected set of signals. Button Clear clears the current state of simulation and returns it to the beginning. Button Help activates the help system where all details concerning the RCS are given.

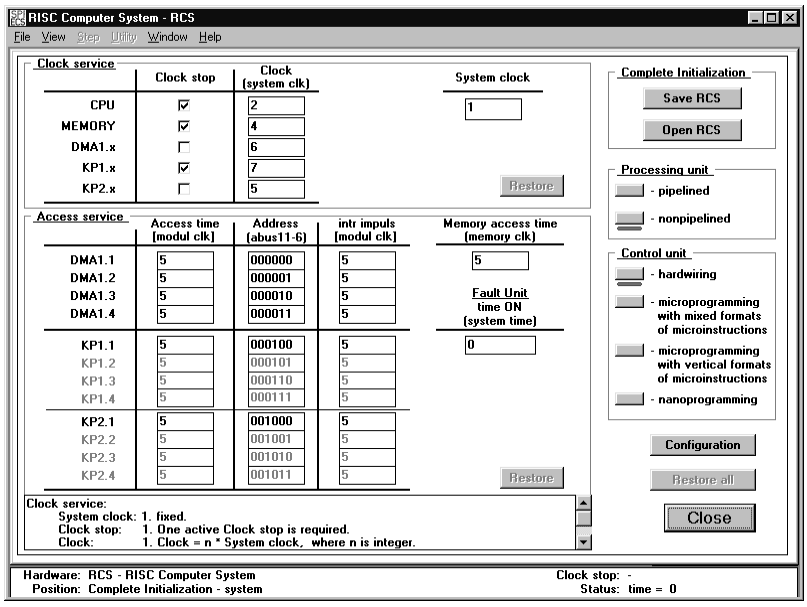

Figure 2. The RCS clock and access times.

The Clock service box allows interactive specification of the System clock and the Clocks for each of the CPU, memory, DMA1.x, KP1.x, and KP2.x. The Clock stop check box enables to select the clocks the occurrence of which should stop the simulation. The Access service box defines the access times, the addresses and the duration of the interrupt pulse for selected peripheral device and DMA controllers, and the memory access time. It allows one to connect or disconnect device FAULT, and define the time when the non-maskable interrupt is raised. The Clock service window provides comments related to the specified clocks. Button Configuration brings back the RCS system configuration screen (Figure 1).

The lower right window, named the Signal graph window, shows the simulation control parameters and clock timing diagrams. Here a user can specify the number of clocks executed until the first stop, the number of instructions executed until the first stop and the duration of program execution until first stop, defined by the number of system clocks. The stops after the specified number of clocks, instructions and duration of the program become valid when the appropriate check boxes are marked. The timing diagrams of the CPU, memory, DMA, KP1 and KP2 clocks are displayed at the moment when the simulation is stopped. The red vertical line points to the parts of the system where the clock(s) occurred, while the next two green lines point to the parts of the system where the next two clock(s) will occur.

The bottom window, named the Status window, shows the status information such as the description of the Block diagram window, the current position inside the RCS, the parts of the system where clock(s) occurred and the total number of system clocks elapsed.

The running of the simulator with the hierarchy of screens of the RCS is described briefly in the following. The first screen with which the simulation begins is the root hierarchical screen, which gives the block structure of the RCS (Figure 3). If one needs more detailed structure of any of the blocks from Figure 3, he/she can move one level down in the hierarchy by positioning the cursor at that block and clicking the mouse button. For example, if the cursor is positioned at the block CPU then the screen giving the block structure of the nonpipelined RISC processor appears (Figure 4). By positioning the cursor at block Register File, one goes one level down in the hierarchy and gets the block structure of the Register File Unit. The same actions applied this time to the GPR block, brings the screen of the last level in the hierarchy (Figure 5). This screen can be obtained directly at any time by clicking on the GPR box in the Hierarchy window. Here one can see the design of this block at the level of standard sequential (registers, flip-flops, etc.) and combinatorial (decoders, logical circuits, etc.) elements. Control signals are generated by the control unit the structure of which can be obtained by clicking the CU box in the Hierarchy window (Figure 6).

At any time during the simulation one can by activating button More examine and set the memory, examine and set CPU, DMA, KP1, and KP2 registers (Figure 7), and get the timing diagram of selected signals from the beginning of the simulation until the current clock period (Figure 8).

The organisation of the simulator with the hierarchical structure of screens makes it possible to carry out the simulation of the functioning of the RCS at various hierarchical levels. At higher levels, one can follow the simulation at the level of signal exchanged between blocks considered here as black boxes. At the lowest level, if it is deemed interesting, one can follow the simulation at the level of registers, flip-flops, logical circuits etc.

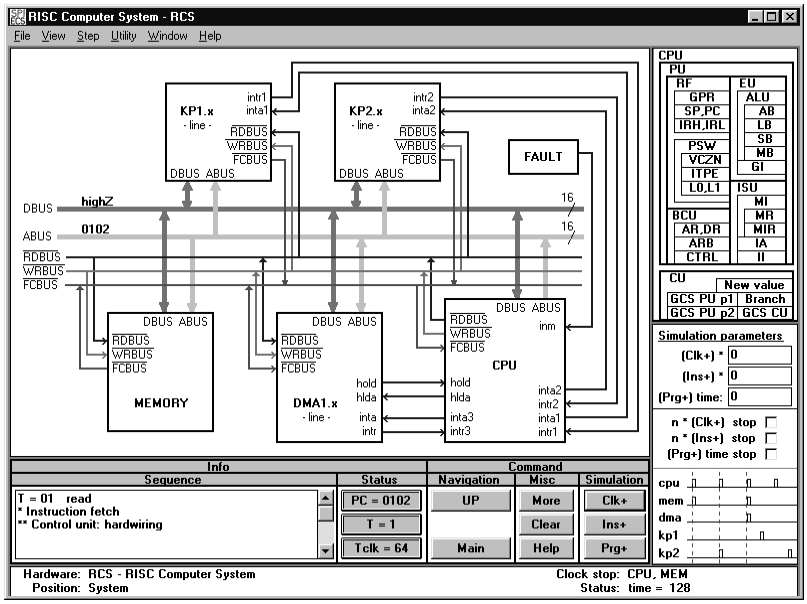

Figure 3. The root hierarchical screen.

This screen shows, at the block level, how the processor (CPU), memory (MEMORY), peripheral device controllers' lines (KP1.x and KP2.x) and DMA controllers' line (DMA1.x) are interconnected through the system bus. Groups of lines, such as data (DBUS) and address bus (ABUS), are given either in green, if they are in the state of high impedance, or in grey with the valid values given in the hexadecimal form, while the control bus lines (RDBUS, WRBUS and FCBUS) are coloured either in blue or red, depending on whether the signal on that line has logical value zero or one, respectively.

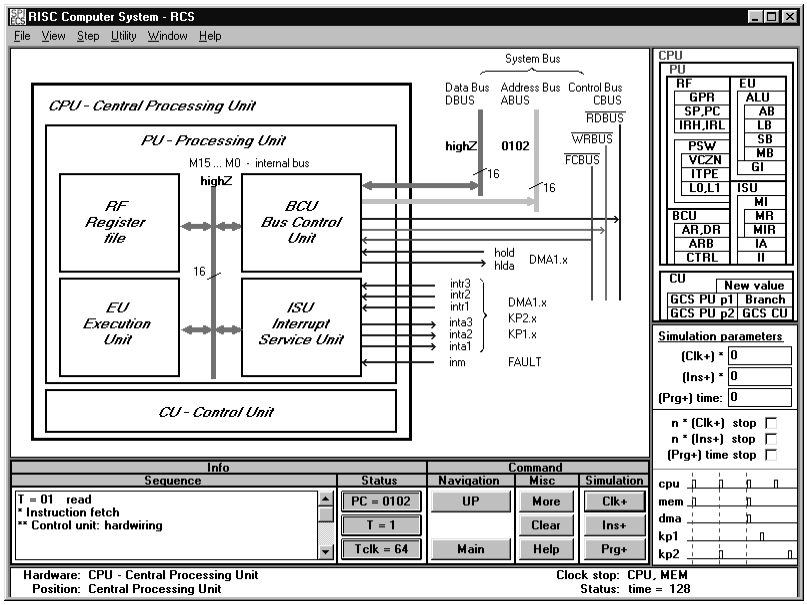

Figure 4. The central processing unit.

The CPU is made up of the Processing unit (PU) and Control unit (CU). The PU contains four units Register File (RF), Execution Unit (EU), Bus Control Unit (BCU) and Interrupt Service Unit (ISU), interconnected by internal bus M. The BCU provides the interface of the CPU with the system bus, while the ISU exchange the interrupt signals with the DMA, KP1.x, KP2.x, and FAULT devices.

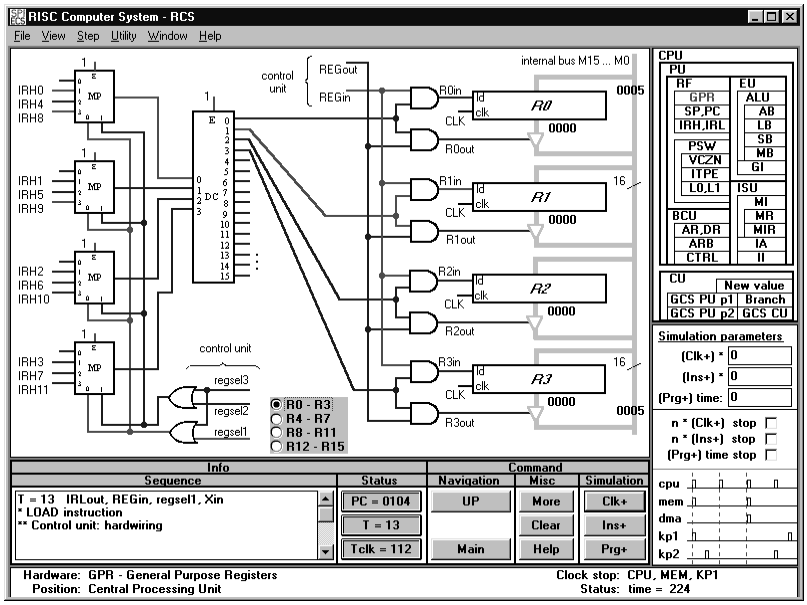

Figure 5. The general purpose registers.

This screen shows the implementation of a group of general-purpose registers including multiplexors (MP), decoder (DC), registers (R0-R3) and logical elements (AND, OR), and the interface to the internal bus (M). The grey box allows one to select the remaining three groups of registers R4-R7, R8-R11, and R12-R15.

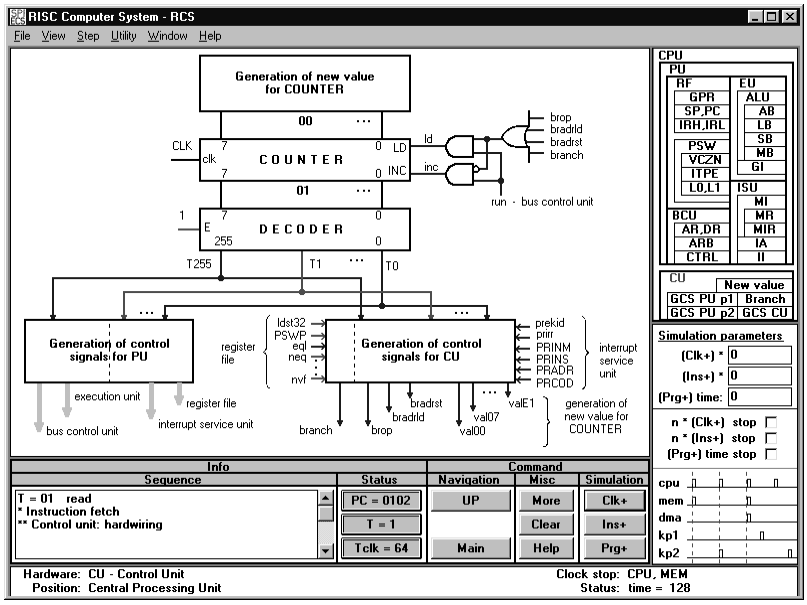

Figure 6. The hardwired control unit.

This screen shows the implementation of the control unit including the counter, the decoder, and blocks Generation of new value for COUNTER (New value), Generation of control signals for PU (GCS PU p1, GCS PU p2), and Generation of control signals for CU (GCS CU, Branch). The New value screen shows the combinatorial circuits, which generate the values to be loaded to the counter. The GCS PU p1 and GCS PU p2 screens show the combinatorial circuits, which generate control signals for the bus control and execution units, and register file and interrupt service units, respectively. The Branch and GCS CU screens show the combinatorial circuits, which generate the branch and the remaining control signals for the control unit, respectively.

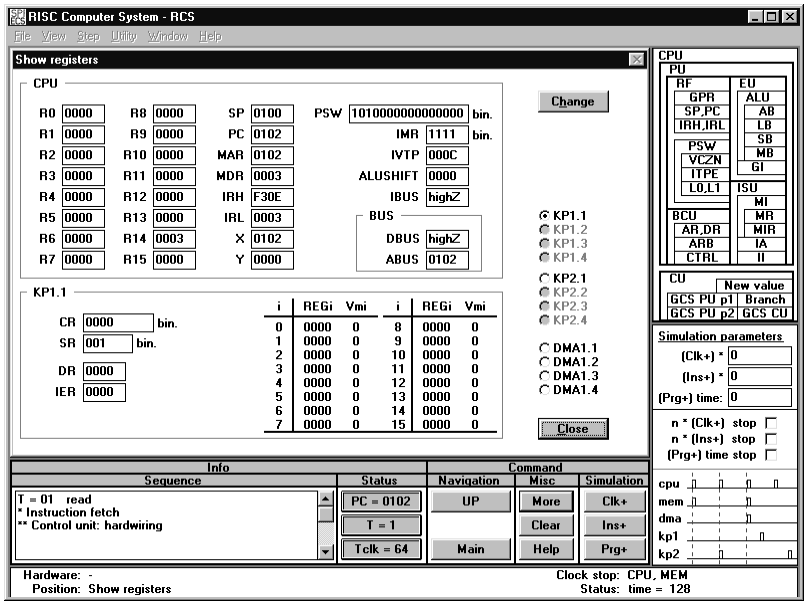

Figure 7. The show registers screen.

This screen shows the contents of the CPU registers and buses, in the upper half, and the contents of the KP1.1 registers with the contents of the simulated peripheral device registers (REGi, i=0-15), in the lower part. If any other KP1.x, KP2.x, or DMA1.x is to be displayed, the appropriate selection box should be marked. The contents of all registers can be manually changed and button Change declares the changes made as valid.

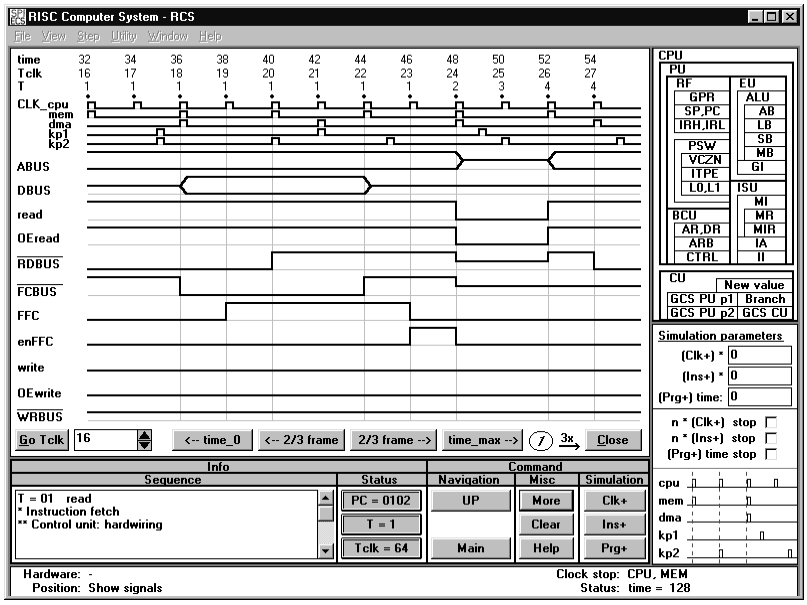

Figure 8. The timing diagrams.

The timing diagram frame contains 12 most frequent clocks (Tclk=16-27). Buttons Go Tclk, ←time_0, ←2/3 frame, 2/3 frame→, time_max→ at the bottom allow one to move the time frame to the desired clock, to the beginning, 8 clocks backward, 8 clocks forward, and to the end, respectively. Any combination of the processor, peripheral device or DMA controllers signals can be selected during the simulator set up.

4. The laboratory experiments and the CALKAS program

The Software Package of the Integrated Educational Computer System is used by the students to carry out laboratory experiments. For each of the above-mentioned courses there is a set of laboratory experiments chosen to cover all major topics lectured in the corresponding course. Each laboratory experiment contains one or more examples depending on the topics it covers. The values used for the initialisation step of the simulator set up are carefully chosen to demonstrate the situations of interest by working with the appropriate simulator of the SPIECS in the way described in section 3. The students are requested to go through all prepared examples at the clock level. The control signals that appear in the information box (Figure 3) point out at the parts of the IECS where an action is going to take place. Based on this, the students should locate relevant parts of the IECS and examine the values of signals at the register transfer level using the SPIECS navigation facilities.

As an aid in the laboratory, a special program named is developed and used by both the instructors and the students [12].

The instructors can initialise the CALKAS program database with the information about all students, for each new class of students. They can also specify the list of laboratory experiments for a particular course. In addition to that, the instructors can enter in the CALKAS program database the list of questions related to the topics covered by the laboratory experiments. Finally, the instructors can get a number of reports about the students' work in the laboratory for a particular class of students. The most typical one is the report which gives for each student separately whether a particular experiment in the laboratory has been carried out or not, then the date and time when the experiment was carried out, and finally whether the students has successfully answered the questions after the completion of the particular laboratory experiment. The instructors can also examine the log file for each laboratory experiment of each student and find out at any time how the student was carrying out each particular experiment. Thus, complete records are being maintained, and based on them decisions are made about those who have passed and failed the work in the laboratory.

The students carry out each laboratory experiment in three steps. The first step required from each student is to register by typing his/her name and password. If the registration is successfully done, the CALKAS program invokes the SPIECS and its top screen appears. The second step is to select and carry out the appropriate laboratory experiment in the way already explained in section 3. The third and final step is the assessment of the students' understanding of the topic covered by the laboratory experiment just completed. For each laboratory experiment, there is a pool of relevant questions from which the CALKAS program randomly generates certain number of questions with offered answers. The student is expected to mark the bullet in front of the answer one regards to be the correct one. The number of questions per each laboratory experiment, the number of answers offered per question, and the period of time within which the answers should be marked are adjustable by the instructor. By activating the appropriate button, the student submits the answers and gets the answer whether he/she has passed or failed. The number of positive answers needed to pass the test is also adjustable by the instructor. The complete interaction of the student with the CALKAS program while performing the above-described three steps is recorded in a log file and becomes part of the student's records.

5. Conclusion

The integrated educational environment, used in the laboratory experiments for the courses in Computer Architecture and Organisation at the Faculty of Electrical Engineering, University of Belgrade, is presented in this paper. It is based on the originally developed methodology for creating educational environments for teaching various topics in the field of computer architecture and organisation. The first step is the logical design of the appropriate part of a computer system down to the register transfer level. The second step is the development of the hierarchical graphical simulator, which allows one to follow the program execution down the clock level in a user-friendly fashion, and the accompanying tools for configuring and initialising of the system.

The integrated educational environment developed has been used in the laboratory for teaching courses in computer architecture and organisation for several years. Based on the number of discussions conducted with the students and the results of the exams, the authors are convinced that the educational environment has been a power aid in teaching and has fulfilled its design objectives. The SPIECS runs under Windows or WindowsNT operating systems, which has made it possible that a lot of students work with it at home.

The related ongoing work goes into two directions. The first one is to make the complete SPIECS a WWW application. The second one is to develop a user-friendly environment that would allow ones to design their own parts of computer systems using the library of standard combinatorial and sequential modules.

6. References

[1] W. Stallings, *Computer Organization and Architecture*, Prentice Hall, New Jersey, 1996.

[2] D. A. Patterson, J. L. Hennessy, *Computer Architecture: A Quantitative Approach*, Morgan Kaufman, San Francisco, California, 1996.

[3] M. Cutler, R. Eckert, "A Microprogrammed Computer Simulator," *IEEE Transactions on Education*, vol. E-30, no. 3, pp. 135-141, August 1987.

[4] E. Pastor, F. Sanchez and A. del Corral, "A Rudimentary Machine: Experiences in the Design of a Pedagogic Computer," *The Workshop on Computer Architecture Education (WCAE-98)*, June 27th, 1998, Barcelona, Spain.

[5] P. Verplaetse and J. Campenhout, "ESCAPE: Environment for the Simulation of Computer Architecture for the Purpose of Education," *The Workshop on Computer Architecture Education (WCAE-98)*, June 27th, 1998, Barcelona, Spain.

[6] Y. Zhang, G. B. Adams III, "An interactive, Visual Simulator for the DLX Pipeline," *IEEE TCCA Newsletter*, September 1997, pp. 9-12.
[7] V. S. Pai, P. Ranganathan, S. V. Adve, "RSIM: An Execution-Driven Simulator for ILP-Based Shared-Memory Multiprocessors and Uniprocessors," *IEEE TCCA Newsletter*, September 1997, pp. 32-38.
[8] C.C. Edmondson-Yurkanan, "Why You should Build a Superscalar Pipeline Simulator," *IEEE TCCA Newsletter*, September 1997, pp. 13-15.
[9] R. N. Ibbett, P. E. Heywood and F. W. Howel, "HASE: A Flexible Toolset for Computer Architects," *Computer Journal*, Vol 38. No 10 pp 775-764, 1995.
[10] Jean-Luc Bechennec, "ASF: A Teaching and Research Object-oriented Simulation Tool for Computer Architecture Design and Performance Evaluation," *The Workshop on Computer Architecture Education (WCAE-98)*, June 27th, 1998, Barcelona, Spain.
[11] J. Djordjevic, A. Milenkovic, N. Grbanovic, M. Bojovic, "An Educational Environment for Teaching a Course in Computer Architecture and Organization," *IEEE TCCA Newsletter*, July 1999, pp. 5-7.
[12] J. Djordjevic, A. Milenkovic, Ivan Todorovic, Darko Marinov, "CALKAS: A Computer Architecture Learning and Knowledge Assessment System," *The Workshop on Computer Architecture Education*, Orlando, Florida, USA, January 1999.