An Integrated Educational Environment for Computer Architecture and Organisation

J. Djordjevic, M. Bojovic, A. Milenkovic
Faculty of Electrical Engineering, University of Belgrade
P.O.Box 3554, 11120 Belgrade, Yugoslavia

Abstract: *The paper presents an integrated educational environment for teaching courses in computer architecture and organisation at the Faculty of Electrical Engineering, University of Belgrade. 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 EVIDAS program. The IECS is developed to demonstrate all major topics lectured in the mentioned courses. The accompanying manuals give all details of its structure. 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. A set of laboratory experiments is devised for students' practical work with the IECS under control of the EVIDAS program. This program is developed to keep the complete evidence about each student's work in the laboratory, assess its knowledge at the end of each laboratory experiment and produce 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. 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 the authors had treated in another paper concerning their first course in Computer architecture and organisation and came to the decision to devise and develop their own educational environment [3, 4]. Based on the number of discussions conducted with the students that have used it in the laboratory experiments and the results of the exams, the authors concluded that the educational environment had been a power aid in teaching the course in Computer architecture and organisation. This resulted in the development of similar environments as help in teaching the second course in Computer architecture and organisation. These educational environments are now being merged creating an integrated educational environment which is the subject of this paper.

The integrated educational environment (the IEE environment), which includes the Integrated Educational Computer System (IECS), the reference manuals for it, and the Software Package of the

IECS (SPIECS), is used by the students to carry out the laboratory experiments. The IECS is devised to cover all topics lectured in the above mentioned courses and its detailed description is given in the accompanying manuals. The IECS is described in Section 2. The SPIECS includes the tools needed to select, configure and initialise the appropriate parts of the IECS, and the graphical simulators of the IECS down to the register transfer level. A brief description of the SPIECS is given in Section 3. A set of laboratory experiments and the EVIT program are devised for students' practical work with the IECS under control of the EVIT program. The laboratory experiments and the EVIDAS program are presented in Section 4. Section 5 contains the conclusion.

2 The integrated educational computer system

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

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

2.1 The CISC processor based educational computer system

The CISC processor based educational computer system (CCS) is made up of the processor, the memory and the input/output subsystem while the communication between them is carried out through the asynchronous bus.

2.1.1 The processor architecture and organisation

The processor program controlled registers include four data, four address, four base and four index registers, and the program counter (PC), the stack pointer (SP), the program status word (PSW), the interrupt mask register (IMR) and the interrupt vector table pointer (IVTP). 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, the register indirect, the memory direct, the memory indirect, the base, the index, the base index, the PC relative, the register indirect with autoincrement and autodecrement and the immediate addressing modes. The instruction set includes the transfer, arithmetic with integer and floating point data types, logic, shift, rotate, control and string instructions. There are internal and external interrupts. The external interrupts are maskable and have assigned priorities.

The processor organisation follows the design approach by which the processor is made up of the processing unit and the control unit. The processing unit is made up of the instruction fetch, decode, operand address calculation and operand reading unit (IFU), four execution units (IEU, FEU, SEU and CEU), the interrupt service unit (ISU) and the bus interface unit (BIU). The IFU unit is responsible for reading and decoding the instruction, calculating the operand address and reading the operand and passing the instruction with the operand to the appropriate execution units. There are four execution units. The IEU unit performs the integer arithmetic, logic and shift operations. The FEU unit carries out the floating point operations. The SEU units is provided for the execution of the string operations. The CEU unit executes program control operations. The ISU unit contains the circuitry necessary to accept all internal and external interrupt requests and establish the address of the appropriate interrupt routine. The $\mathbf{intr_0}$ till $\mathbf{intr_3}$ and $\mathbf{inta_0}$ till $\mathbf{inta_3}$ lines are used to exchange the interrupt request and interrupt acknowledgment signals between the processor and input/output units, respectively. The BIU unit contains the circuitry necessary to exchange the bus request and bus acknowledgment signals with the bus arbiter and perform the read, write and interrupt vector number acquisition bus cycles on the bus. 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.

2.1.2 The memory

The memory of the educational computer system has the capacity of 64Kbytess, the memory word length is 8 bits and the processor generated addresses are for 8 bit words. The memory module has the usual input address lines, the bidirectional data lines and the input control lines to start the memory

read or write operations and the output control line to indicate that the requested memory cycle has been finished.

2.1.3 The input/output subsystem

The input/output subsystem is made up of three input/output units and one direct memory access controller. Each input/output unit is made up of a peripheral device controller and the peripheral device itself. The control part of the peripheral device controller is responsible to accept data from the input peripheral device, store into its data register, set the ready bit in its status register and generate an interrupt request if the interrupt enable bit in its control register is set. In addition to that, it reset the ready bit in its status register when the contents of the data register is read by the processor. The control part of the peripheral device controller also reset the ready bit in its status register when the contents of the data register is written by the processor. In addition to that, it sends the contents of its data register to the output peripheral device, set the ready bit in its status register and generate an interrupt request if the interrupt enable bit in its control register is set. The peripheral device controller has not only the usual control, status and data registers, but, also, the interrupt vector register where the interrupt vector table entry number of this input/output unit is being kept.

The DMA controller is assigned to one of the input/output units. In addition to the usual control, status and data registers, it, also, has the source and destination address registers, the block count register and the interrupt vector register. It supports, not only transfers between the peripheral device and memory and vice versa, but, also, the memory to memory transfers. The transfers can be carried out in the cycle stealing and burst modes. The direct memory access controller is a slave when its registers are being initialised by the processor and a master during the transfer of data. The required mode of operation is specified by writing the appropriate value into the control register of the direct memory address controller.

The input/output address space is memory mapped, so that the highest 8Kbytes addresses of the memory address space are reserved for the purpose of addressing the registers in the peripheral device controllers and the direct memory access controller.

2.1.4 The bus

The components of the educational computer system are interconnected through the asynchronous bus. The bus is made up of the 16 address lines, 8 data lines and three control lines. By generating the active values on the rd (read) or wr (write) control lines, the processor or the direct memory access controller, as the bus masters, specify that the read or write cycle should be performed on the bus. By generating the active values on the fc (function completed) control line, the memory, the peripheral device controller or the direct memory access controller, as the bus slaves, specify that the initiated read or write cycle has been completed. Any bus cycle is preceded by the arbitration. Therefore, the processor and the DMA controller, as the possible bus masters, are connected with the bus arbiter with the private pairs of lines. The bus request line (breq) is used by the possible master to send a bus request to the arbiter, while the bus grant line (bgrant) is used by the arbiter to grant the use of the bus to the possible master.

2.2 The RISC processor based educational computer system

The RISC processor based educational computer system (RCS) is made up of the processor, the memory and the input/output subsystem while the communication between them is carried out through the asynchronous bus.

The processor architecture is the load/store type. Thus, only the Load and Store instructions can access operands in the memory, while all remaining instructions work with operands in one of the general purpose registers. The processor program controlled registers include, besides 16 general purpose registers, the program counter (PC), the stack pointer (SP), the program status word (PSW), the interrupt mask register (IMR) and the interrupt vector table pointer (IVTP). Data types supported are 16-bit signed and unsigned integers. The length of the instructions is 16 or 32 bits. The instruction format is the variable one, so that, depending on the operation specified by a particular instruction, the three address, the two address, the one address and the zero address instruction formats are used. The Load and Store instructions can only explicitly specify the use of one of the following addressing

modes: the immediate (only for Load), the memory direct, the register indirect and the register indirect with displacement. 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. The external interrupts are maskable and have assigned priorities. The **intr$_0$** till **intr$_3$** and **inta$_0$** till **inta$_3$** lines are used to exchange the interrupt request and interrupt acknowledgment signals between the processor and input/output units, respectively.

For this processor architecture two types of processor organisation have been implemented the nonpipelined one and the pipelined one.

The nonpipelined processor organisation follows the design approach by which the processor is made up of the processing unit and the controlling unit. The processing unit is made up of the register file unit (RFU), the execution unit (EXU), the interrupt service unit (ISU) and the bus interface unit (BIU) interconnected with the 16-bit internal bus. The register file unit contains the program controlled and auxiliary registers. The execution unit is made of the ALUSHIFT block, where the basic arithmetic, logic and shift operations are performed, the auxiliary X and Y registers, where the input data for the ALUSHIFT block are kept, and the combinational circuit, where the values to be set into the condition code bits of the program status register are generated. The bus interface unit contains the circuitry necessary to perform the read, write and interrupt vector number acquisition bus cycles on the bus and arbitrate between the processor and the DMA controller requests for accessing the bus. The **hold** and **hlda** lines are used to exchange the bus request and bus acknowledgment signals between the processor and the DMA controller. The interrupt service unit contains the circuitry necessary to accept all internal and external interrupt requests and establish the address of the appropriate interrupt routine. The adopted processor design approach, where the processor is made up of a separate processing and controlling unit, made it possible to design four types of controlling units for the same processing unit. 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 follows the design approach by which the processor is made up of five stages each one for a separate phase of an instruction execution. These processor stages and instruction phases are: the instruction fetch (IF), the instruction decode and operands read (ID), the operation execution (IE), the memory access (MEM) and the result write (WR). The IF stage is provided for reading an instruction. The ID stage is used for decoding an instruction and reading two operands from the register file. The EX stage performs the execution of arithmetic, logic and shift operations, the calculation of the memory address for the load and store operations and the branch address calculation for the control operations. The MEM stage carries out the memory reads and writes for the load and store instructions, respectively, and the PC register modification for the control instructions. The WR stage writes the result into the register file. The structural, data and control hazards are reduced to a minimum. The structural hazards could have occurred when the main memory and the register file accesses were made. In order to avoid them separate instruction and data caches are implemented, and the register file was implemented with such elements which allow two registers to be read and one to be written at the same time. The read after write data hazard was eliminated by using the data forwarding technique. The control hazards are minimised by using the dynamic prediction mechanism with the branch target buffer. The interrupt mechanism was not implemented here, since its implementation would complicate the processor design to such an extent that the elements of the pipeline mechanism could not be followed easily.

The structures of the memory, the input/output subsystem and the bus are very similar to the ones of the **C**ISC processor based educational system and are not considered here.

2.3 The educational hierarchical memory system

The educational hierarchical memory system (HMS) 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. In addition to that, the hierarchical memory system was envisaged to be such that all most commonly used techniques for realising the

virtual memory and translation lookaside buffer, the cache memory and the interleaved memory could be demonstrated. Finally, the aim was to develop a user friendly environment which would make it possible to follow the above process at the clock level and examine, at any time, the values of all signals of the hierarchical memory system down to the register transfer level. 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.

2.3.1 The virtual memory and translation lookaside buffer

The virtual memory and translation lookaside buffer (TLB) are demonstrated by using three types of lookaside buffer and three types of virtual memory. The TLB holds certain number of descriptors of most frequently accessed pages or segments. For any address translation request, coming from the processor, the TLB checks, according to the mapping technique implemented, whether the descriptor is in it. If the descriptor is in the TLB, the virtual to real address translation is carried out and the real address returned to the processor. In the case of the TLBs for segmented and segment paged virtual memories the access and address violation checks is performed. If any or both checks fail, the TLB generates an interrupt. If the descriptor is not in the TLB, it goes into the appropriate segment or/and page table(s) and checks the appropriate descriptor. If the segment or page is in the main memory, its descriptor is loaded in the appropriate entry in the TLB. Now, for the same address translation request the TLB check is carried out again. The activities are now the same as for the above explained case when the descriptor is in the TLB. If the segment or page is not in the main memory, the TLB generates an interrupt. The TLB entry to be replaced with a new descriptor is selected according to the Least Recently Used (LRU) algorithm for the TLB with the associative mapping and according to either the LRU or First In First Out (FIFO) algorithm for the TLB with the set associative mapping.

The processor sending address translation requests to the TLB is simulated. The HMS environment allows a user to initialise a table in the simulated processor with address translation requests and time intervals between them before the simulation starts. Once the simulation is started, the simulated processor according to the information obtained from the table will generate address translation requests.

The virtual memory is realised by interrelated activities of the TLB and the operating system. Therefore, some of the operating system activities dealing with situations when the processor and the TLB are switched from a process to a process, when an interrupt is generated by the TLB etc. are also simulated. Here again the HMS environment allows a user to specify parameters relevant for the operating system activities in such situations.

2.3.2 The cache memory

The cache memory is demonstrated by using three types of cache memory realised with the direct, associative and set associative mapping. The cache memory holds certain number of most frequently accessed blocks from the main memory. For any main memory access request, coming from the processor, the cache memory checks, according to the mapping technique implemented, whether the block is in it. If the block is in the cache memory, the requested access is carried with the cache memory, and, in the case of the read access request, the data read is returned to the processor. If the block is not in the cache memory, the block is transferred from the main memory into the cache memory. For the same memory access request the cache memory check is carried out again. The activities are now the same as for the above explained case when the block is in the cache memory. The cache memory block to be replaced with a new block is selected according to the First In First Out (FIFO) algorithm for the cache memories with the associative and set associative mapping.

Some other activities in the cache memory depend on whether the write back or store through technique for updating the main memory is used. The cache memory with the direct mapping uses the write back technique. As a consequence, the cache memory block, selected to be replaced with a new block, is first returned to the main memory, and then the new block is transferred from the main memory into the cache memory. However, this is done only if at least one write access for the cache

memory block to be replaced has been carried out. The cache memory with the associative mapping uses the store through technique. Therefore, there is no need to return to the main memory the block selected to be replaced with a new block. The cache memory with the set associative mapping uses the write back technique with buffering. The cache memory block, selected to be replaced with a new block, is returned to the main memory only if at least one write access for that cache memory block has been carried out. However, this block is first written only into a buffer block, then the new block is transferred from the main memory into the cache memory, and, finally, the block selected to be replaced is transferred from the buffer block into the main memory.

The processor is realised in a similar way as for the virtual memory.

2.3.3 The interleaved main memory

The interleaved main memory is demonstrated in a system made up 16 units, 16 memory modules, a split transactions synchronous bus and the arbiter.

Each unit can be configured to generate either the single word accesses or the block accesses. The single word accesses are typical for a processor fetching instructions or accessing scalar values or a direct memory access controller working with a slow peripheral device. The block accesses are typical for a processor transferring blocks between the cache memory and the main memory or a direct memory access controller working in the burst mode with a fast peripheral device. A unit includes the bus interface and the requester. The bus interface includes all circuitry necessary to perform the appropriate operations when the unit is either a master or a slave. When the unit is a master, it sends a bus request to the arbiter and performs a read request or write request cycle when it gets the grant. When the unit is a slave, it accepts data, requested by an earlier read request cycle, in a data available cycle. 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. It is realised in a similar way as the processor for the virtual and cache memories.

A memory module contains the bus interface and the RAM memory. The bus interface contains all circuitry necessary to perform the appropriate operations when the unit is either a slave or a master. When the module is a slave, it accepts a write or read request from the bus and initiates the write or read operation in the RAM module. As the result of the read operation performed in the RAM module, the bus interface obtains data requested by a bus read request cycle, and starts with activities as a bus master. It sends a bus request to the arbiter and performs a data available cycle when it gets the grant. The RAM memory of the module is used to store data. The HMS environment allows a user to initialise locations of interest in the simulated RAM memory and specify the desired access time for each memory module separately. Each memory module can obtain its number in the range from 0 to 15. In addition to that, five possible ways of interleaving memory modules can be specified. One of them is consecutive addresses in the same module, the other one is consecutive addresses in 16 consecutive modules and the remaining three are possible cases of mixtures of the first two.

The bus is realised as a split transactions synchronous bus. The bus cycles performed are the write request, the read request and the data available. In order to make it possible for the module to return data to the unit that initiated a read request cycle, the unit sends its identifier with the read request cycle. When the data requested are read, the module as the master uses the identifier in the data available cycle to send it to the appropriate unit.

The arbiter realises the parallel arbitration between all units and modules. Each of them is connected with the arbiter with a pair of lines. One of them is used for sending a request to the arbiter, and the other one is used for receiving a grant from the arbiter. The memory modules are given higher priority than the units.

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 to select the simulator of one of three educational computer systems (Fig.1). 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.

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. The CCS system does not require to be configured. The RCS system requires to specify whether the processor with the nonpipelined or the pipelined organisation is to be used. If the nonpipelined processor is specified, one has to select which of four implemented control units, such as the ones with the hardwiring, microprogramming with the mixed and vertical formats of microinstructions and nanoprogramming, is to be used with the processing unit. For the HMS system simulators of the virtual memory and TLB and the cache memory there is no need to configure them. For the interleaved memory one must specify one of five possible ways of interleaving memory modules, assign numbers to modules, define for each unit whether it is the one with the single word or the block access and assign identifiers to units.

The second step of the simulator set up is to initialise the selected simulator. In the case of the CCS and RCS systems it consists of the initialisation of the processor, the memory and the input/output units. The initialisation of the processor involves the loading of the PC register with the starting address of the program, the IVTP register with the starting address of the interrupt vector table, the IMR register with the value specifying the input/output units from which interrupts will be accepted, the SP register denoting the top of the stack, the 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. The initialisation of the input/output units means 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. In the case of the HMS system the initialisation depends on the simulator selected. In the case of the virtual memory it includes the initialisation of the table of the simulated processor, some registers in the TLB, and parts of the main memory containing the page or/and segment tables. In the case of the cache memory it includes the initialisation of the table of the simulated processor, some registers in the cache memory and parts of the main memory accessed by the cache memory. For the interleaved main memory it includes the loading of the table of the simulated requester and the memory module locations accessed by the units.

The selected simulator set up can be carried our either partially (the Partial Set up button) or completely (the Complete Set up button) as shown in Fig. 2.

If the partial simulator set up is chosen one has to go through a number of steps first to configure and second to initialise the simulator as required. The initialisation of the simulator is carried out partially for parts of the educational computer system such as processor registers, memory locations, peripheral devices etc. Each of the simulator set up steps can be done either interactively or from files. If the interactive simulator set up is chosen, one can interactively write, display and change the configuration and values of all parts of the educational computer system. Here one can also write his own programs in a symbolic manner, translate, link and load them into the memory using for this purpose devised tools such as the editor, the translator, the linker and the loader. Each of the simulator set up steps carried out interactively can be saved in separate files. These files can be used for the partial simulator set up from files. There is also a possibility to save in a file a complete simulator set up carried out partially.

The complete simulator set up, which includes both the configuring and the initialisation of the simulator, can be carried out either from a file or from test examples (Fig. 3). If the complete set up from a file is to be done one has to type the file name. If the complete set up from a test example is to be done one has to select the appropriate test example. It should be noted that the test examples are also simulator set up files. They are prepared by the instructors to illustrate during the simulator run

some typical situations lectured in the courses in Computer architecture and organisation. In both cases the files are prepared using the partial simulator set up. The complete simulator set up can be also done from a file into which the state of the simulation was previously saved if the simulation was interrupted before being completed.

The partial simulator set up and the complete simulator set up from file are predominantly used by instructors to carefully prepare laboratory experiments. The complete simulator set up from test examples is normally used by students to carry out laboratory experiments.

When the simulator set up step is completed the simulator run step can be carried out. This can be either the start of a completely new simulation (the Start button) or the resumption of an earlier interrupted simulation (the Resume button).

3.2 The simulator run

The simulator run is the step performed in the same way for the CCS system, the RCS system and HMS system of the IECS system regardless of how the simulator set up was done. Therefore the basic features of the simulator run are given in the following using one of them, namely the RCS system.

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. Due to the fact that a limited number of elements can be displayed on a screen, a hierarchical scheme of the RCS, as shown in Fig. 4, is developed. 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.

Each screen, in general, is made up of two windows. The larger window in the upper part of the screen, named the Block diagram window, contains either only a composition of combinational and sequential circuits, if this is a leaf block in the hierarchical scheme, or a composition of subblocks, that can be further selected, and combinational and sequential circuits, if this is not a leaf block in the hierarchical scheme. The smaller window in the lower part of the screen, named the Info and Command window, is divided into the Info window at its left hand side and Command window at its right hand side. The Status buttons PC and Tclk in the Info window display the current values of program counter and the number of clock periods executed, respectively. The Info window Sequence gives either the value of the microprogram counter mPC, in case that one of the microprogram Control units are used, or the value of the step counter, in case that the hardwired Control unit is used, 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 Command window contains three groups of command buttons: Navigation, Simulation and Miscellaneous. The Navigation command buttons are UP and Hierarchy. Button UP allows to move from the current screen to the screen one level up in the hierarchy, while button Hierarchy allows to move directly from the current screen to the ECS Hierarchy screen (Fig.4). Only from the ECS Hierarchy screen one can go directly to the screen of any of the blocks down in the hierarchy. This can be achieved by positioning the cursor at the appropriate block and clicking the mouse button. The Simulation command buttons Clk+, Ins+ and Prg+ allow to continue with the simulation just with one clock period or with the number of clock periods required to complete the current instruction or the complete program, respectively. The Miscellaneous command buttons are Show, Clear and Help. Button Show opens the window which allows 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 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 functioning of the educational computer system and its simulator are available.

The simulator run with the hierarchy of screens of the ECS is described briefly in the following. The first screen with which the simulation begins is the one with the block structure of the educational computer system (Fig. 5). One can also arrive to that screen from the ECS Hierarchy screen (Fig. 4) by positioning the cursor at the ECS block and clicking the mouse button. This screen shows at the block level how the processor (CPU), memory (MEM), DMA controller (DMA) and peripheral devices (PER1, PER2 and PER3) are interconnected through data (DBUS), address (ABUS) and control (CBUS (RDBUS, WRBUS and FCBUS)) lines of the system bus. The simulation can be carried out at

this hierarchical level by activating either Clk+, Ins+ or Prg+ button. What one can follow, then, are the values of signals exchanged at the block level. The values for groups of lines, such as for data (DBUS) and address bus (ABUS) lines, are given in the hexadecimal form, while single lines are colored either in blue or red depending on whether the signal on that line has logical value zero or one, respectively. If one needs more detailed structure of any of the blocks from Fig. 5., he can move one level down in the hierarchy by positioning the cursor at that block and clicking the mouse button. As an example of this one can assume that the cursor is positioned at block CPU on the screen given in Fig. 5. and the mouse button clicked. What appears is the screen giving the block structure of the educational processor (Fig. 6). From this screen one can go one level down and get more detailed structure of any of four units of the Processing Unit (Register File Unit, Executing Unit, Bus Control Unit or Interrupt Service Unit) and the Control Unit. By positioning the cursor at block Register File Unit and clicking the mouse button, one goes one level down in the hierarchy and gets the block structure of the Register File Unit (Fig. 7). The same actions applied this time to the GPR block of screen from Fig. 7., brings the screen of last level in the hierarchy (Fig. 8). Here one can see the design of this block at the level of standard sequential (registers, flip-flops, etc,) and combinational (decoders, logical circuits, etc.) elements.

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

If one find useful to examine and set registers, he can use button Show at any moment during the simulation and get the screen as shown in Fig. 9. By using the same button one can get the timing diagram of selected signals from the beginning of the simulation until the current clock period in the form shown in Fig. 10. for the signals relevant for the realisation of the read cycle on the system bus.

4 The laboratory experiments and the EVIDAS program

The Software Package of the Integrated Educational Computer System (SPIECS) is used by the students to carry out laboratory experiments under control of the EVIDAS program.

For each of the above mentioned courses there is a set of laboratory experiments carefully 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 for each laboratory experiment. The aim is 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 (Fig. 4) 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.

Complete work in the laboratory is carried out under control of the specially developed program, named the EVIDAS program, used by both the instructors and the students.

The instructors can initialise the EVIDAS 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 EVIDAS 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 made about those who have passed and failed the work in the laboratory.

The students carry out each laboratory experiment under control of the EVIDAS program in three steps. The first step required from each student is to register by typing its name and passport. If the registration is successfully done, the EVIDAS program invokes the SPIECS and its top screen appears (Fig.11). The second step is to select and carry out the appropriate laboratory experiment in the way already explained in section 3.1. If one, for example, wants to carry out a laboratory experiment with the RISC processor based computer system the appropriate button from the screen given in Fig. 1. should be activated and the screen given in Fig. 2. appears. Now the Complete initialisation button should be activated and the screen given in Fig. 3. appears. Here one of test examples should be selected. By activating the OK button one returns back to the screen given in Fig. 2. Finally, from here by activating the Start button the simulation starts and the top screen, given in Fig. 5., appears. From here the simulation is performed in the way described in section 3.2. 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 EVIDAS program randomly generates certain number of questions. Each question has a few 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 its answers and as a result of this it gets the answer whether he 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 EVIDAS 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 (IEE), 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 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 EVIDAS program. The IECS is originally designed by the authors with the aim to incorporate all major topics lectured in the mentioned courses. The SPIECS, developed to demonstrate the functioning of the IECS at the register transfer level, and the EVIDAS program, used for students' evidence and assessment, run on both the Windows95 and WindowsNT operating systems.

The integrated educational environment is based on the originally developed and already used methodology for creating educational environments for teaching various topics in the field of computer architecture and organisation. The first step in the methodology is the logical design of the appropriate part of a computer system down to the register transfer level. The next step is the development of the graphical simulator with accompanying tools which allow one to configure and initialise the system and carry out the simulation during the execution of the test examples. The simulation is done at the clock level in a user friendly fashion.

The environments developed with the described methodology have been used for teaching courses in computer architecture and organisation for several years. They have been useful aids and favourably accepted by the students. The authors feel that the students are rigidly lead in this environment through the predetermined situations leaving little room for their initiative and creativity. Therefore, the related ongoing research is directed towards the development of a user friendly environment that would allow ones to design their own parts of computer systems using the library of standard combinational 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] J. Djordjevic, A. Milenkovic, N. Grbanovic, M. Bojovic, "*An Educational Environment for Teaching a Course in Computer Architecture and Organization*," Proceedings of the IEEE/ACM HPCA-98 Workshop on Computer Architecture Education, Las Vegas, NV, January 1998.

[4] J. Djordjevic, A. Milenkovic, S. Prodanovic "*A Hierarchical Memory System Environment*," Proceedings of the IEEE/ACM ISCA-98 Workshop on Computer Architecture Education, Barcelona, Spain, June 1998.

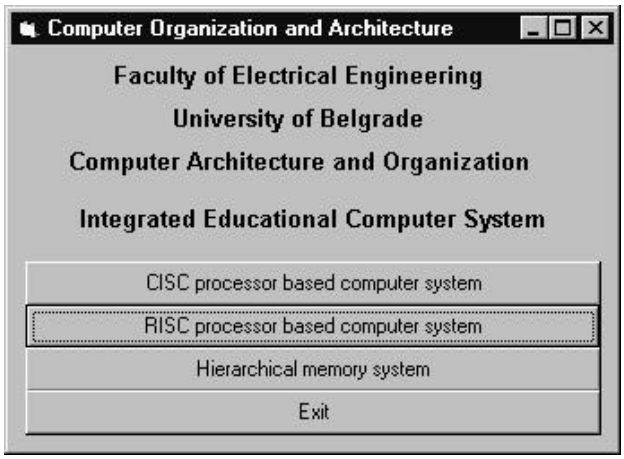

Fig.1 The simulator selection

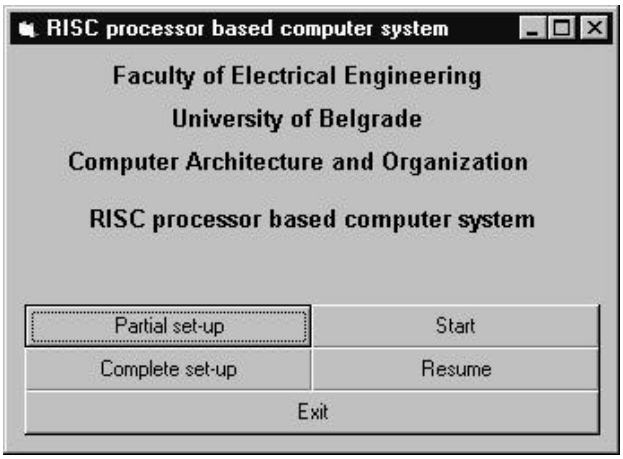

Fig. 2 The simulator set up

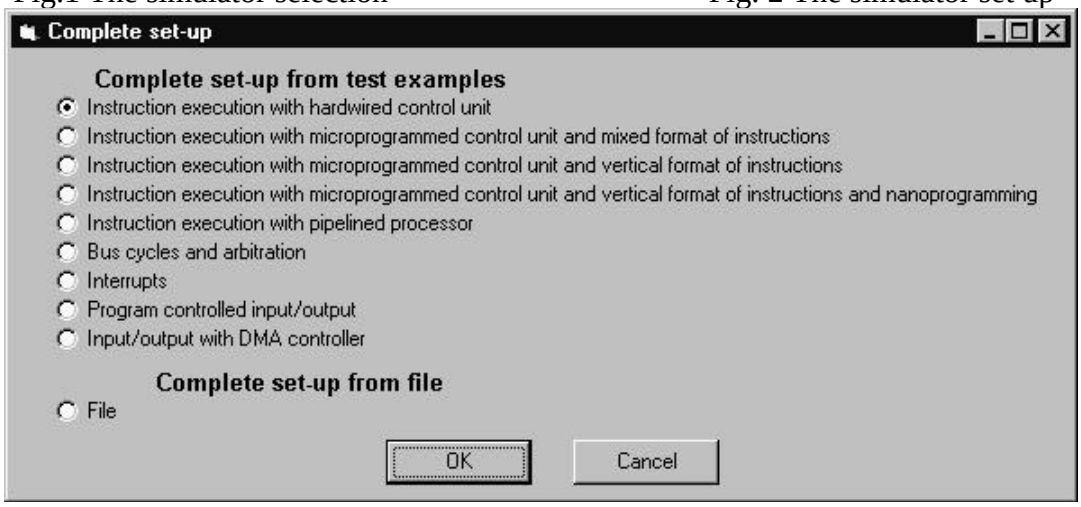

Figure 3 The complete simulator set up

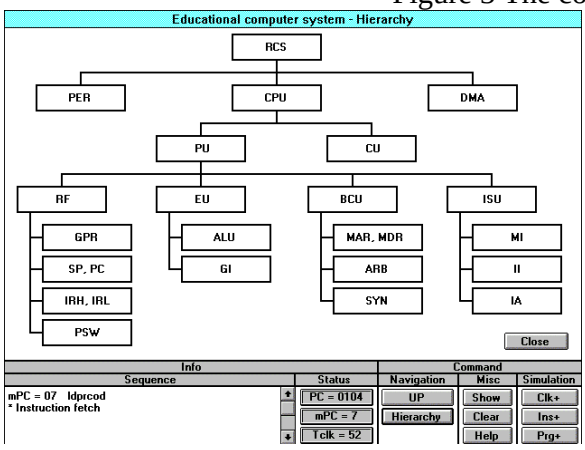

Figure 4. The RCS Hierarchy screen

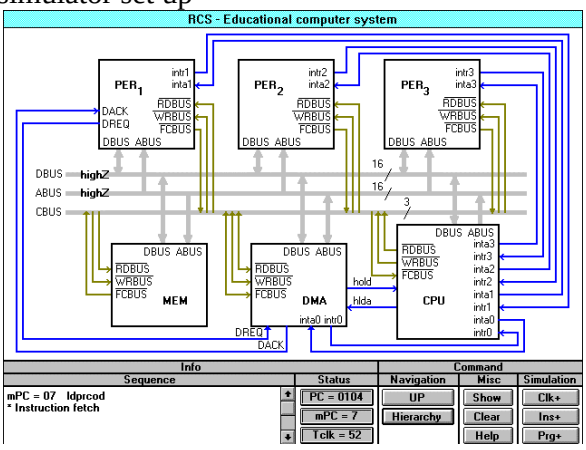

Figure 5. The RCS screen

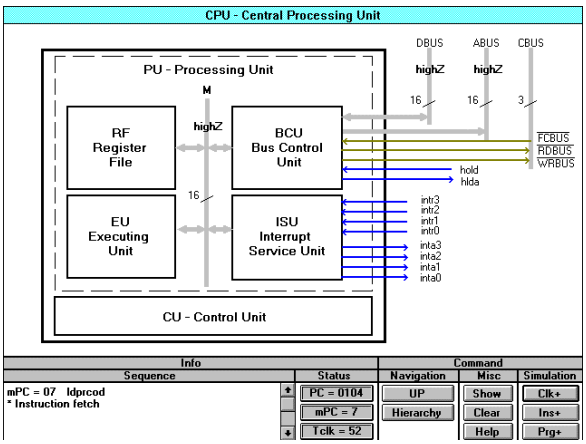

Figure 6. The central processing unit screen

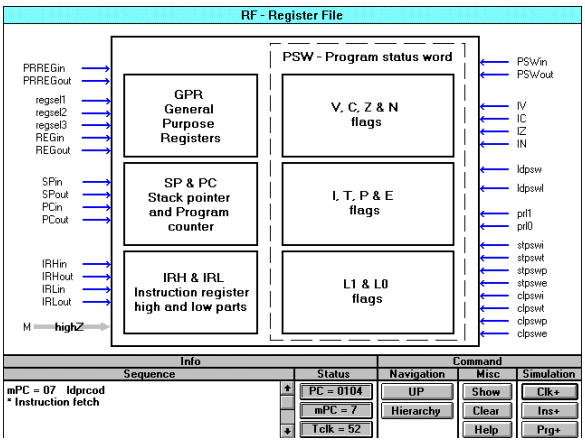

Figure 7. The Register File screen

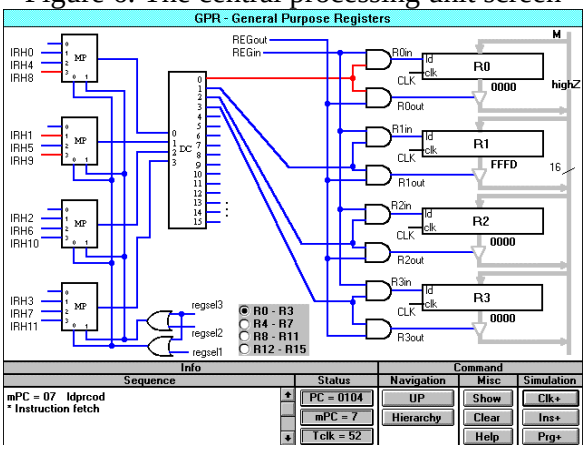

Figure 8. The General Purpose Registers screen

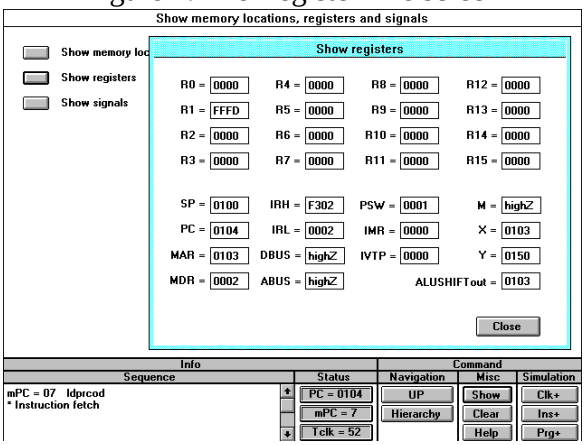

Figure 9. The show registers screen (foreground) and the show memory locations, registers and signals screen (background)

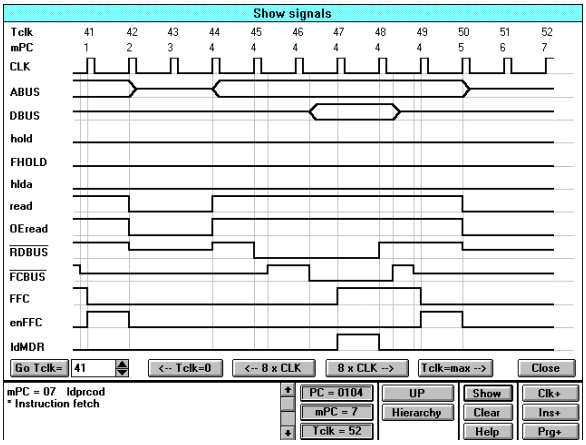

Figure 10. The show signals screen

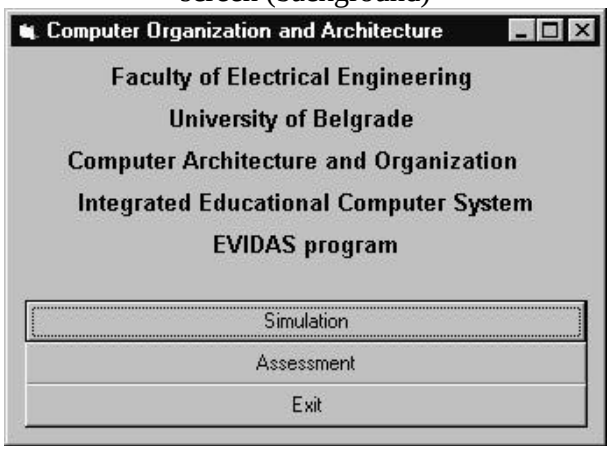

Figure 11. The EVIDAS screen