An Integrated Environment for Teaching Computer Architecture

To bridge the gap between learning and doing in traditional computer architecture courses, students require a different educational approach, such as the integrated environment the authors describe.

Jovan Djordjevic
Aleksandar Milenkovic
Nenad Grbanovic
University of Belgrade

A major problem in teaching computer architecture and organization courses is how to help students make the cognitive leap that connects their theoretical knowledge with practical experience. Numerous researchers involved in computer architecture and organization education have tackled this problem, resulting in a variety of educational tools for computer system simulation. The tools differ greatly in scope, target architecture complexity, simulation level, and user interface, as the "Related work" sidebar on p. 70 explains.

The available educational systems vary in how they handle digital system simulation. They usually offer tools for creating hardware component libraries, viewing simulation results, and conducting statistical analysis of system performance. Available systems range from sophisticated ones, for complex analysis, to simpler ones that are more readily understood by users, both instructors and students.

Beyond system simulation, an educational system should support three key objectives. First, it must cover an extensive range of computer architecture and organization topics. Second, it should graphically depict a computer system, from the block level to the register-transfer level. Third, it must provide the means to follow system functions at the program, instruction, and clock cycle levels.

Because our analysis of available tools and systems produced none that met the stated objectives, we developed the integrated educational environment (so named because of its multifaceted capabilities) described in this article. We use this environment in two courses at the University of Belgrade. The first course, a second-year undergraduate course, covers basic computer structure concepts, including processor, memory, input/output subsystem, and bus.[1] The second course goes one step further and covers, for example, CISC and RISC processor architecture and organization, and organization of pipelined processors, storage, interconnections, and memory.[2]

Integrated environment

Our environment consists of several major components:

- the Integrated Educational Computer System (IECS) plus related reference manuals,
- the IECS software package (SPIECS), which runs under Microsoft Windows or WindowsNT,

0272-1732/00/$10.00 © 2000 IEEE

Table 1. Computer architecture and organization topics that an educational environment system should address.

Processor				
Architecture	Organization	Memory	Bus	Input/output
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 controller implementations

- a set of laboratory experiments, and
- the Computer Architecture Learning and Knowledge Assessment System (CALKAS).

The IECS, the authors' original design, covers all the topics outlined in Table 1. SPIECS supplies the software tools to select, configure, and initialize the IECS, including graphical simulators. The laboratory experiments enable students to simulate practical work with the IECS, and the CALKAS program helps instructors evaluate the students' work.

Hardware

The IECS portion of the environment encompasses three self-contained systems:

- RISC-processor-based computer system (RCS),
- CISC-processor-based computer system (CCS), and
- hierarchical memory system (HMS).

For both RCS and CCS, structures of the memory, input/output subsystem, and bus are similar, with memory capacities of 128 Kbytes and 64 Kbytes, respectively. The input/output subsystem supports 12 peripheral devices: 8 with peripheral device controllers and 4 with direct memory access (DMA) controllers. An asynchronous bus interconnects the components.[3]

RCS and CCS differ primarily in processor architecture and organization.

RCS architecture

RCS has a load/store processor architecture with general-purpose registers and registers such as the program counter, stack pointer, and so on. RCS supports 16-bit signed and unsigned integer data types. It has a three-address instruction format with a 32-bit instruction length. The load/store instructions' addressing modes are immediate, memory direct, register indirect, and register indirect with displacement. All remaining instructions implicitly use the register-direct-addressing mode. The instruction set includes the transfer, arithmetic, logic, shift, rotate, and control instructions. Interrupts are both internal and external.

We implemented RCS as both a nonpipelined and a pipelined processor. The nonpipelined organization contains the processing and control units. The processing unit consists of the register file, execution unit, interrupt service unit, and bus interface unit, which interconnects with the 16-bit internal bus. For the processing unit, we designed four types of control units. One is hardwired; the others are microprogrammed with the mixed and vertical formats of microinstructions, as well as nanoprogramming.

A five-stage organization characterizes the pipelined processor, and each stage corresponds to a separate instruction-execution phase. These processor stages and instruction phases are the instruction fetch, instruction decode and operand read, operation execution, memory access, and result write. To

Related work

One group of educational tools centers on rudimentary architectures with fewer than 20 instructions.[1,2] The tools support functional-level simulation, which enables students to follow the processor dataflow graphically. The educational systems described by Verplaetse and Campenhout[3] as well as by Zhang[4] center on the DLX instruction set.[5]

Verplaetse and Campenhout present a system that supports microcode or pipelined organizations. Each provides a window showing the system structure, register states, and signal lines. Users can view and edit memory, and view pipeline activity.

Zhang's simulator provides animated versions of the key figures and tables from Patterson and Hennessy.[5] Users follow pipeline activity details at the clock or instruction level.

In a second category of tools, several powerful simulators demonstrate commercial processor functions. These include the SparcV8[6] and PowerPC601,[7] both of which provide rich instruction sets, numerous addressing modes and data types, and high simulation speeds. In addition, these simulators can run Unix programs, C compilers, and SPEC92 benchmarks. The primary goals of these simulators are as-fast-as-possible simulations of target systems, statistical analysis, and time diagram viewing. Visualization, however, is either nonexistent or rudimentary at best.

In a third category is the HASE system,[8] which is based on an object-oriented database management system that runs in an X-Windows/Motif environment. This system's aim is high-level simulation and visualization. Users have access to the read-only HASE component library and can also create their own component libraries. Users create the system architecture to be simulated via a graphical program in which system components are selected, placed, and linked. During the simulation, the HASE system creates an event trace file that contains all required data about the current simulation run. The program that animates the simulation, or the program for viewing time diagrams, can use this file as an input.

Bechennec[9] describes a system similar to the HASE. Users develop a C++ application that enables system block creation, system configuration, and simulation runs. Available classes include those for blocks, links, data transfers, data classes, and simulators. With these classes, users create and link blocks, define data transfer rules, and finally configure the whole system. Textual reports show simulation results, including parameter values.

References

1. M. Cutler and R. Eckert, "A Microprogrammed Computer Simulator," *IEEE Trans. Education*, Vol. E-30, No. 3, Aug. 1987, pp. 135-141.
2. E. Pastor, F. Sanchez, and A. del Corral, "A Rudimentary Machine: Experiences in the Design of a Pedagogic Computer," *IEEE TCCA Newsletter* , February, 1999, pp. 51-53.
3. P. Verplaetse and J. Campenhout, "ESCAPE: Environment for the Simulation of Computer Architecture for the Purpose of Education," *IEEE TCCA Newsletter*, February 1999, pp. 57-59.
4. Y. Zhang and G.B. Adams III, "An Interactive, Visual Simulator for the DLX Pipeline," *IEEE TCCA Newsletter*, Sept. 1997, pp. 9-12.
5. D.A. Patterson and J.L. Hennessy, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, San Francisco, Calif., 1996.
6. V.S. Pai, P. Ranganathan, and S.V. Adve, "RSIM: An Execution-Driven Simulator for ILP-Based Shared-Memory Multiprocessors and Uniprocessors," *IEEE TCCA Newsletter*, Sept. 1997, pp. 32-38.
7. C.C. Edmondson-Yurkanan, "Why You Should Build a Superscalar Pipeline Simulator," *IEEE TCCA Newsletter*, Sept. 1997, pp. 13-15.
8. R.N. Ibbett, P.E. Heywood, and F.W. Howel, "HASE: A Flexible Toolset for Computer Architects," *Computer J.*, Vol. 38, No. 10, 1995, pp. 775-764.
9. J.-L. Bechennec, "ASF: A Teaching and Research Object-Oriented Simulation Tool for Computer Architecture Design and Performance Evaluation," *Proc. Workshop Computer Architecture Education* (WCAE-98), IEEE Computer Society Press, Los Alamitos, Calif., 1998.

avoid structural hazards, RCS contains separate instruction and data caches, and the register file that simultaneously allows two registers to be read and one to be written. Data forwarding eliminates read-after-write data hazards, and dynamic prediction with the branch target buffer minimizes control hazards.

CCS architecture

The CCS processor architecture features separate data, address, base, and index registers, plus the standard program counter, stack pointer, and program status word registers, among others. The processor, which supports 8-bit signed and unsigned integer data types, 16-bit floating-point numbers, and character strings, executes one-address instructions in lengths up to 4 bytes. These instructions include transfer, arithmetic with integer and floating-point data types, logic, shift, rotate, control, loop control, and string. Ten addressing modes are supported: register direct, register indirect, memory direct, memory indirect, base, index, base index, relative, register indirect with auto increment and auto decrement, and immediate. Finally, interrupts are both internal and external.

Eight units comprise the CCS processor: the instruction fetch, decode, operand address calculation, and operand reading unit; five execution units (integer, floating-point, string, control, and loop control); interrupt service unit; and bus interface unit. Each unit has its own processing and control.

HMS

The HMS comprises the virtual memory and translation looka-side buffer (TLB), cache memory, and main memory. Initially, we planned to develop an HMS in which the process of accessing the TLB, the cache memory, and the interleaved memory could be realized altogether. Although this environment was feasible, such a system would be too complex and difficult for students' use. Therefore, we split the HMS environment into three separate entities: virtual memory and TLB, cache memory, and interleaved memory.

Virtual memory and TLB. The virtual memory comprises the paged, segmented, and segment-paged organizations, and includes the TLB with the direct, associative, and set-associative mappings. The TLB typically receives the virtual address, accesses the appropriate descriptor, performs the access rights and bounds checks, and generates the real address. Occasionally, the TLB brings a new descriptor from the segment and/or page table and replaces a TLB entry with the descriptor according to the LRU and FIFO algorithms, when appropriate.

Cache memory. The cache memory supports the direct, associative, and set-associative mappings. It receives a main memory request, checks the cache memory for a hit, and performs the read or write, according to the specific mapping technique's features. The cache memory implements the FIFO and LRU replacement algorithms where needed and uses both write-back and write-through main memory updating techniques. It implements block transfers from the main memory in case of a cache miss, including the block-buffering, critical-word-first, and early-processor-start techniques.

Interleaved main memory. This part of the system consists of 16 units, 16 memory modules, the bus, and the arbiter. Users can configure each unit to generate either the single-word or the block accesses. Each unit includes the bus interface and the requester. The bus interface contains all circuitry for the appropriate operations when the unit is either a master or a slave. The requester simulates parts of a processor or a DMA controller, which generates the memory reads and writes, and accepts data returned in response to a read.

A memory module consists of the bus interface, which contains all circuitry for the appropriate operations when the unit is either a slave or a master, and the RAM. Users can specify five ways of interleaving memory modules. We implemented the split-transactions synchronous bus and the parallel arbiter.

Software

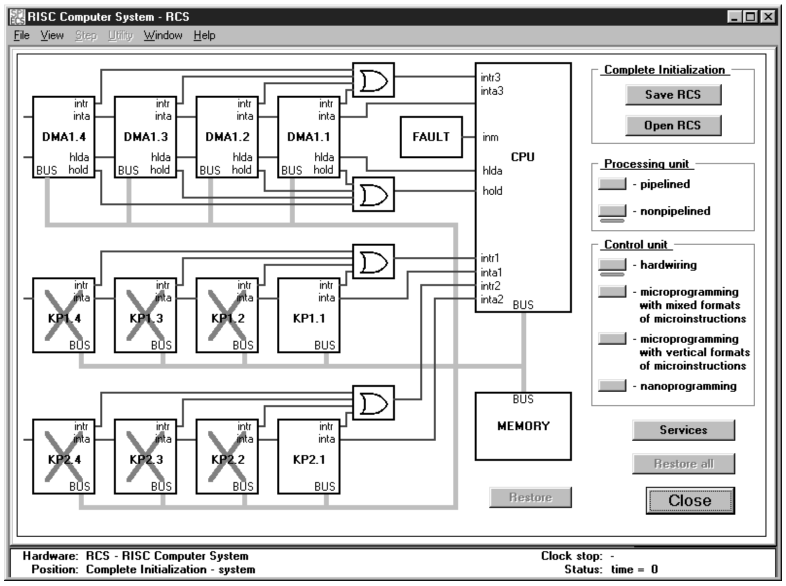
SPIECS includes the graphical simulators of the IECS and the tools for selecting, setting up, and running them. Users must perform three steps in working with this software. First, they select the simulator of either the RCS, CCS, or HMS. If they select the HMS, they must also select one of three simulators for the virtual memory and TLB, cache memory, and interleaved memory. Further, users must, for the virtual memory and cache memory, select one of three types of TLBs and cache memories. Next, they set up the simulator; then run the simulator.

Setting up the simulator

Simulator setup is similar for all three systems. For simplicity, we use the RCS as our example throughout. Although our intent is for instructors to use the setup procedure in preparing laboratory experiments, students can also perform the procedure. Ideally, this would occur much later in their course work.

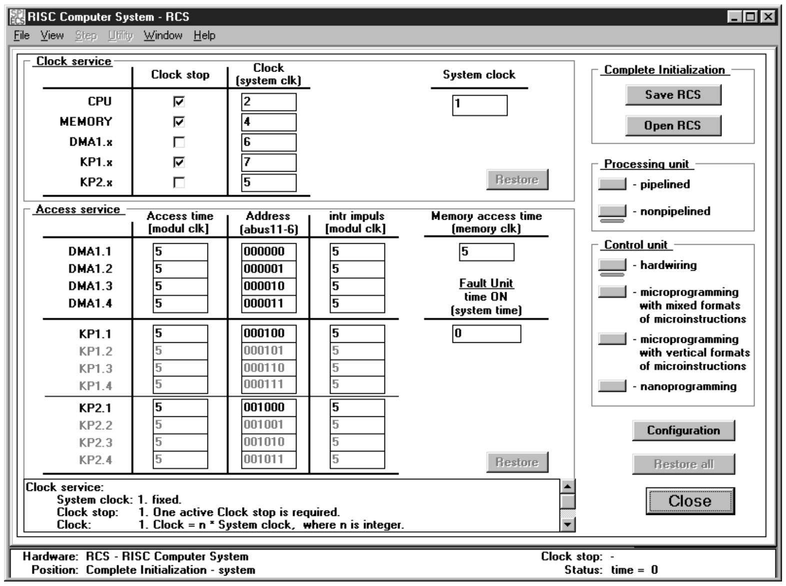
The first step in setting up an RCS system simulator is configuration. The user chooses the number of peripheral devices and DMA controllers, selects processing and control units via the screen shown in Figure 1a (next page), and defines clock and access times via the screen in Figure 1b.

Initialization of the simulator encompasses initialization of the processor, memory, and peripheral device and DMA controllers. Processor initialization involves loading of the programmable registers. Similarly, memory initialization includes loading the appropri-


RISC Computer System - RCS
File View Step Utility Window Help
intr inta DMA1.4 hlda hold BUS
intr inta DMA1.3 hlda hold BUS
intr inta DMA1.2 hlda hold BUS
intr inta DMA1.1 hlda hold BUS
FAULT
intr3 inta3 inm CPU hlda hold intr1 inta1 intr2 inta2 BUS
intr inta KP1.4 BUS
intr inta KP1.3 BUS
intr inta KP1.2 BUS
intr inta KP1.1 BUS
intr inta KP2.4 BUS
intr inta KP2.3 BUS
intr inta KP2.2 BUS
intr inta KP2.1 BUS
BUS MEMORY
Complete Initialization
Save RCS
Open RCS
Processing unit
- pipelined
- nonpipelined
Control unit
- hardwiring
- microprogramming with mixed formats of microinstructions
- microprogramming with vertical formats of microinstructions
- nanoprogramming
Services
Restore all
Restore
Close

Hardware: RCS - RISC Computer System
Position: Complete Initialization - system
Clock stop: -
Status: time = 0

(a)


RISC Computer System - RCS
File View Step Utility Window Help
Complete Initialization
Save RCS
Open RCS
Processing unit
- pipelined
- nonpipelined
Control unit
- hardwiring
- microprogramming with mixed formats of microinstructions
- microprogramming with vertical formats of microinstructions
- nanoprogramming
Restore
Configuration
Restore all
Close

Clock service

	Clock stop	Clock (system clk)
CPU	☑	2
MEMORY	☑	4
DMA1.x	☐	6
KP1.x	☑	7
KP2.x	☐	5

System clock: 1

Access service

	Access time [modul clk]	Address (abus11-6)	intr impuls [modul clk]
DMA1.1	5	000000	5
DMA1.2	5	000001	5
DMA1.3	5	000010	5
DMA1.4	5	000011	5
KP1.1	5	000100	5
KP1.2	5	000101	5
KP1.3	5	000110	5
KP1.4	5	000111	5
KP2.1	5	001000	5
KP2.2	5	001001	5
KP2.3	5	001010	5
KP2.4	5	001011	5

Memory access time (memory clk): 5

Fault Unit time ON (system time): 0

```
Clock service:
      System clock: 1. fixed.
      Clock stop:    1. One active Clock stop is required.
      Clock:         1. Clock = n * System clock,  where n is integer.
```

Hardware: RCS - RISC Computer System
Position: Complete Initialization - system
Clock stop: -
Status: time = 0

(b)

Figure 1. RCS system configuration (a) and RCS clock and access times (b).

ate memory locations with values either interactively entered or obtained as the result of using the assembler, linker, and loader. To initialize the peripheral device and DMA controllers, the user loads the simulated input peripheral device with the data and time sequence that this device will generate.

Users can choose either partial or complete simulator setup. If partial, they must separately initialize each part of the system. A complete setup initializes the whole system all at once.

Figure 1a shows the RCS screen for configuring the I/O subsystem, and the processing and control units. The I/O subsystem can contain up to four DMA controllers, denoted as DMA1.*x*. The subsystem can contain eight peripheral device controllers, denoted as KP1.*x* and KP2.*x*, in which *x* = 1,…,4. Each controller can be disconnected or reconnected by consecutive mouse clicks on the appropriate box.

The Complete Initialization box at the top right-hand corner of the screen includes buttons (Open RCS and Save RCS) that let users configure and initialize the whole system from, or save the current state into, a file. The next box lets users choose a pipelined or a nonpipelined processing unit implementation. If nonpipelined is specified, users must select the control unit to be used with the processing unit, from the Control unit box.

Restore buttons on the lower right-hand side of the screen let users cancel some or all interactively made changes, respectively. The Services button lets users call up the clock and access times screen shown in Figure 1b.

The Clock service box at the top of the screen shown in Figure 1b lets users interactively specify the system clock and the clocks for the CPU, memory, DMA1.*x*, KP1.*x*, and KP2.*x*. The Clock stop check box lets users select which clocks should stop the simulation.

The Access service box, below the Clock service box, lets users define access times, addresses, and duration of the interrupt pulse for selected peripheral device and DMA controllers. It also lets them specify the memory access time; connect or disconnect device FAULT, and define when it should raise the nonmaskable interrupt. The Configuration button at the lower right-hand side of the window returns users to the RCS configuration screen of Figure 1a.

The Clock service window at the bottom of the screen provides comments related to the specified clocks.

Running the simulator

By running the simulator, users can follow the RCS's signal values during program execution after a clock, an instruction, or a complete program.

Figure 2 shows the root of the hierarchical scheme we developed for the RCS. The large block diagram window occupies most of the screen. If the user is simulating a nonleaf block, this window contains a composition of subblocks, and combinatorial and sequential circuits. The user can further select each subblock until reaching the leaf block, which contains only combinatorial and sequential circuits.

The window at the upper right of the screen shows the CPU's hierarchy. The user can directly reach any level in the hierarchy by selecting the appropriate box in this window.

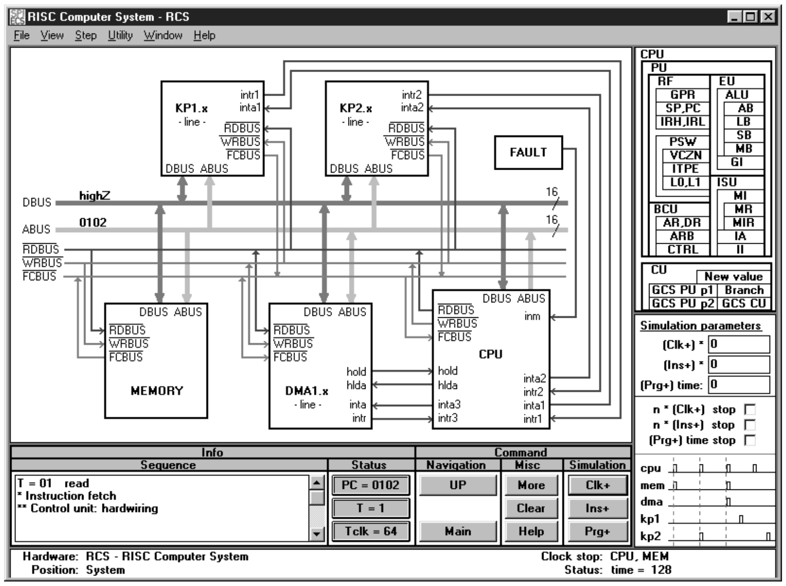
For convenience, any screen can be reached in two ways. The first one is by clicking at the subblocks in the block diagram window until the design is reached. The second one is to click the corresponding box in the hierarchy window.

The Info window at the bottom contains the Sequence window and the Status buttons. The Sequence window shows the microprogram or step counter's value and the control signals generated for that clock period. It briefly explains the actions to occur during that clock period. The Status PC, T, and Tclk buttons display the current values of the program counter and the step counter, and the number of the CPU clock periods executed.

The Command window contains three groups of command buttons: Navigation, Miscellaneous, and Simulation. The UP command button in Navigation lets users move up one level from the current screen, while Main returns to the root hierarchical screen of Figure 2.

Of the Miscellaneous command buttons, More opens the window that allows users to examine and set the values of memory locations, processor and peripheral device, and DMA controller's registers. This window also lets users draw the timing diagrams of a selected set of signals. Clear resets the current simulation state and returns the simulation to the initial state. Help activates the RCS help system.

The Simulation command button Clk+ lets users continue the simulation until the first clock appears. The Ins+ and Prg+ buttons do the same with the number of clock periods required to complete the current instruction or program.



T = 01 read
* Instruction fetch
** Control unit: hardwiring

PC = 0102
T = 1
Tclk = 64

Hardware: RCS - RISC Computer System
Position: System
Clock stop: CPU, MEM
Status: time = 128

Figure 2. The root hierarchical screen shows, at the block level, how the processor, memory, peripheral device controllers' lines (KP1.*x* and KP2.*x*), and DMA controllers' line (DMA1.*x*) interconnect through the system bus. The data (DBUS) and address bus (ABUS), appear in green if they are in the state of high impedance; otherwise they appear gray with hexadecimal values. The control bus lines (RDBUS, WRBUS, and FCBUS) appear in blue if the signal has a logical value of zero; and red, if the value is one.

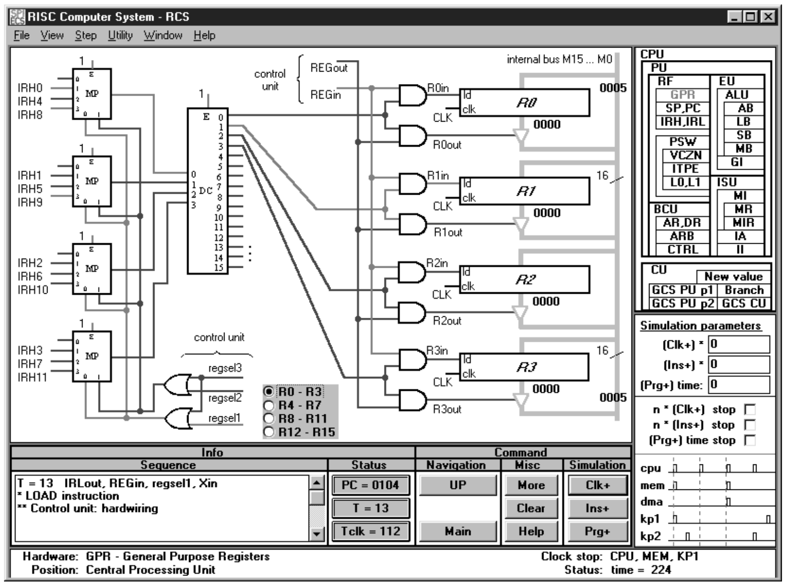
The signal graph window in the screen's lower right shows simulation control parameters and clock-timing diagrams. Here users specify the number of clocks and instructions executed until the first stop. The stops become valid when the user marks the appropriate check boxes.

The signal graph window also displays the timing diagrams of the CPU, memory, DMA, KP1, and KP2 clocks when the simulation stops. The first vertical line points to the system parts where the clocks occurred, while the next two lines point to where the next two clocks will occur.

The status window at the very bottom of the screen shows several key parameters: what the block diagram window shows, the parts of the system where the clocks occurred, and the total number of elapsed system clocks.

Simulation process walk-through

The simulation begins with the root hierarchical screen, which in our example depicts the RCS block structure, as Figure 2 shows. If users need a more detailed structure of any block,



(Clk+) * 0
(Ins+) * 0
(Prg+) time: 0

n * (Clk+) stop ☐
n * (Ins+) stop ☐
(Prg+) time stop ☐

[✓ R0 - R3] R4 - R7 R8 - R11 R12 - R15

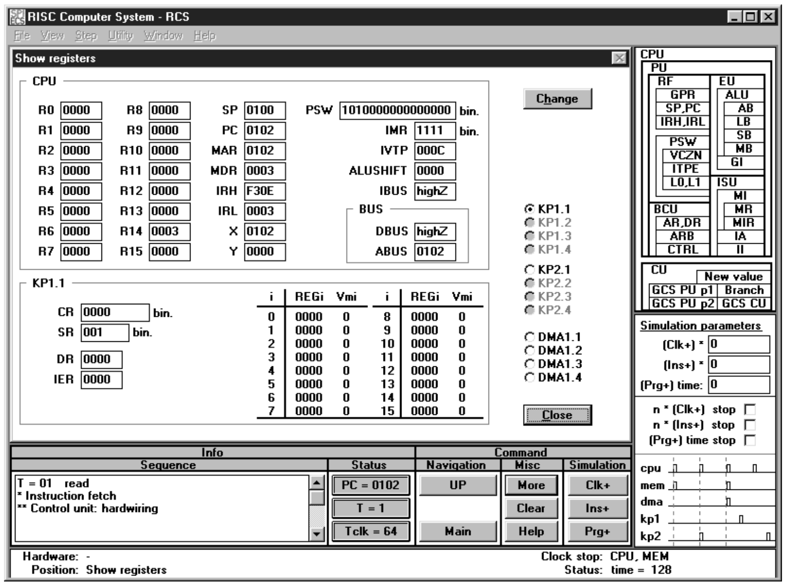
T = 13   IRLout, REGin, regsel1, Xin
* LOAD instruction
** Control unit: hardwiring

PC = 0104
T = 13
Tclk = 112

Hardware: GPR - General Purpose Registers
Position: Central Processing Unit

Clock stop: CPU, MEM, KP1
Status: time = 224

Figure 3. The general-purpose registers screen shows the implementation of GPRs including multiplexers (MP), decoder (DC), registers (R0-R3), and logical elements, and the interface to the internal bus (M15...M0).



CPU

R0	0000	R8	0000	SP	0100	PSW	1010000000000000 bin.		
R1	0000	R9	0000	PC	0102	IMR	1111 bin.		
R2	0000	R10	0000	MAR	0102	IVTP	000C		
R3	0000	R11	0000	MDR	0003	ALUSHIFT	0000		
R4	0000	R12	0000	IRH	F30E	IBUS	highZ		
R5	0000	R13	0000	IRL	0003				
R6	0000	R14	0003	X	0102	DBUS	highZ		
R7	0000	R15	0000	Y	0000	ABUS	0102		

BUS

[✓ KP1.1] KP1.2 KP1.3 KP1.4
KP2.1 KP2.2 KP2.3 KP2.4
DMA1.1 DMA1.2 DMA1.3 DMA1.4

KP1.1

CR 0000 bin.
SR 001 bin.
DR 0000
IER 0000

i	REGi	Vmi	i	REGi	Vmi
0	0000	0	8	0000	0
1	0000	0	9	0000	0
2	0000	0	10	0000	0
3	0000	0	11	0000	0
4	0000	0	12	0000	0
5	0000	0	13	0000	0
6	0000	0	14	0000	0
7	0000	0	15	0000	0

(Clk+) * 0
(Ins+) * 0
(Prg+) time: 0

n * (Clk+) stop ☐
n * (Ins+) stop ☐
(Prg+) time stop ☐

T = 01   read
* Instruction fetch
** Control unit: hardwiring

PC = 0102
T = 1
Tclk = 64

Hardware: -
Position: Show registers

Clock stop: CPU, MEM
Status: time = 128

Figure 4. The show registers screen displays 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.

they can move one level down in the hierarchy by pointing and clicking the mouse at the appropriate block. For example, if the cursor is positioned at the block CPU in the block diagram window, the resulting screen shows the block structure of the nonpipelined RISC processor with subblocks, one of them being the Register File (RF). By positioning the cursor at the RF subblock, users move down one level in the hierarchy, and the resulting screen shows the RF unit block structure. Similarly, the same actions applied to the general-purpose registers (GPR) subblock calls up the screen of the hierarchy's last level, as Figure 3 shows.

This screen, which can be obtained directly by clicking on the GPR box in the hierarchy window, shows the block design at the level of standard sequential elements. Elements include registers and flip-flops, and combinatorial elements, such as decoders and logical circuits.

Users click on the More button to call up the show registers screen to examine and set the memory, CPU, DMA, KP1, and KP2 registers, as Figure 4 shows. Similarly, users would click More to get the timing diagram of selected signals from the beginning of the simulation until the current clock period, as Figure 5 shows.

Experiments

Essentially, experiments are designed to help students perform simulation scenarios in which they apply what they have learned to real-world sorts of situations but in a laboratory setting. Accordingly, instructors devise experiments with the SPEICS software. The first and second courses comprise five and ten laboratory experiments, respectively.

A laboratory experiment contains one or more examples, depending on the topic. For example, an experiment on interrupts includes examples that illustrate the jump to the interrupt routine, the return from the interrupt routine, interrupt masking, nesting, priority, and so on.

In preparation, instructors carefully choose values for the simulator setup's initialization step to demonstrate real-world situations of interest to students. Students then run the simulator to execute all prepared examples. At simulation stops, students examine the signal values of combinatorial and sequential circuits with the simulator's graphical facilities. This helps students grasp a computer system's inner workings.

Assessment

The last component of our environment, the Computer Architecture Learning and Knowledge Assessment System (CALKAS) program, helps instructors evaluate students' progress.[4]

Instructors initialize the CALKAS program database with information about students, such as the student ID and name, and also specify the laboratory experiments for a particular course. Additionally, instructors enter questions, related to the topics covered by the laboratory experiments, into the database. Finally, instructors obtain reports about the students' laboratory work.

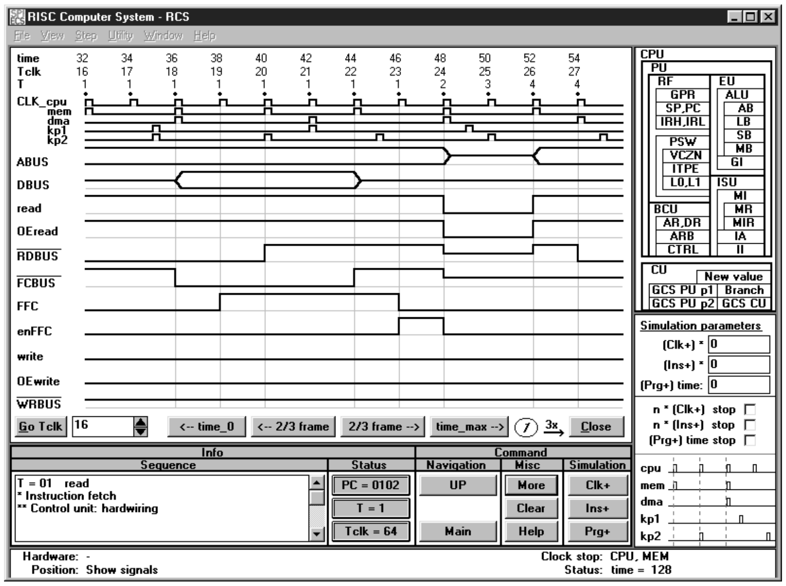
Typically, instructors request a report that indicates if an individual student has carried out a particular laboratory experiment. This report includes the date and time of the experiment, and indicates if the student has successfully answered the experiment's associated questions.

Instructors can also examine the log file for each laboratory experiment of each student and learn how the student performed each particular experiment. The CALKAS program thereby helps us maintain complete records of student achievement.

For their part, students follow three steps in carrying out the experiments. First, students register by entering their name and password. If the registration is successful, the CALKAS program invokes the SPIECS, which lets students perform the second step: to select and carry out the appropriate laboratory experiment.

The third step is the assessment of the students' understanding of the topic covered by the laboratory experiment. For each experiment, the CALKAS program randomly generates a number of questions with multiple-choice answers from an existing database. Students then take the test and, by activating the appropriate button, submit their answers then learn immediately whether they passed or failed.

Instructors can modify the number of test questions and answers, as well as the amount of time students have to answer them and the number of positive answers needed to pass the test. The students' complete interaction with CALKAS is recorded in a log file and becomes part of the students' records.



time 32 34 36 38 40 42 44 46 48 50 52 54
Tclk 16 17 18 19 20 21 22 23 24 25 26 27
T 1 1 1 1 1 1 1 1 2 3 4 4

Simulation parameters
(Clk+) * 0
(Ins+) * 0
(Prg+) time: 0
n * (Clk+) stop
n * (Ins+) stop
(Prg+) time stop

T = 01 read
* Instruction fetch
** Control unit: hardwiring

PC = 0102
T = 1
Tclk = 64

Hardware: -
Position: Show signals
Clock stop: CPU, MEM
Status: time = 128

Figure 5. The timing diagrams screen contains the 12 most frequent clocks (Tclk = 16–27). Buttons near the bottom of the screen (Go Tclk and so forth) move the time frame to the desired clock. Users can select any combination of the processor, peripheral device, or DMA controllers signals.

We have used the integrated educational environment for five years. On the basis of exam results and discussions with students, the educational environment has fulfilled its design objectives in two ways. First, the average grade has roughly improved 20%. Second, the students show deeper understanding of the topics lectured. The software has enabled many students to work at home and prepare for working in the laboratory.

Our ongoing work is two-pronged. First, we want to make the SPIECS a Web-based application, which will foster distance education and eliminate the crowding in the labs. Second, we want to develop a user-friendly environment that lets students design their own computer systems by using the library of standard combinatorial and sequential modules. MICRO

References

1. W. Stallings, *Computer Organization and Architecture*, Prentice Hall, Upper Saddle River, N.J., 1996.
2. D.A. Patterson and J.L. Hennessy, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, San Francisco, Calif., 1996.
3. J. Djordjevic et al., "An Educational Environment for Teaching a Course in Computer Architecture and Organization," *IEEE TCCA Newsletter*, July 1999, pp. 5-7.
4. J. Djordjevic et al., "CALKAS: A Computer

Architecture Learning and Knowledge Assessment System," *Proc. Workshop Computer Architecture Education* (WCAE-99), IEEE Computer Soc. Press, Los Alamitos, Calif., 1999.

Jovan Djordjevic is an associate professor in the Department of Computer Engineering, Faculty of Electrical Engineering, at the University of Belgrade. His research interests include computer architecture, parallel computer systems, and hardware description languages. Djordjevic received a BS in electrical engineering from the University of Belgrade, and an MSc and a PhD in computer science from the University of Manchester.

Aleksandar Milenkovic is an assistant professor in the Department of Computer Engineering, Faculty of Electrical Engineering, at the University of Belgrade. His research interests include computer architecture, parallel computer systems, computer-aided design, and the Internet. Milenkovic received a BS, an MS, and a PhD in computer engineering from the University of Belgrade. He is a member of the IEEE and the IEEE Computer Society.

Nenad Grbanovic is a graduate student at the Department of Computer Engineering, Faculty of Electrical Engineering, at the University of Belgrade. His research interests include systems programming, database systems, and hardware description languages. Grbanovic received a BS in electrical engineering from the University of Belgrade.

Direct comments about this article to Aleksandar Milenkovic at the Faculty of Electrical Eng., Univ. of Belgrade, PO Box 35-54, 11120 Belgrade, Serbia, Yugoslavia; milenkovic@kiklop.etf.bg.ac.yu.