FACTA UNIVERSITATIS (NIŠ)
SER.: ELEC. ENERG. vol. 18, No. 2, April 2005, 237-252

Simulators for Courses in Advance Computer Architecture

Anastas Mišev, Marjan Gušev

Abstract:
The usage of simulator in teaching computer architecture courses has proven to be the most acceptable way, especially when the simulators offer rich graphical and visual representation of the architecture. In this paper we present several simulators used to teach ILP (Instruction Level of Parallelism) courses. The simulators cover wide area of concepts such as internal logic organization, datapath, control, memory behavior, register renaming, branch prediction, and overall out of order execution. Special dedicated simulators cover details in internal organization like Tomasulo approach and scoreboard for organization of reservation stations. This innovative approach in laboratory exercises is used for advanced ILP course.

Keywords: computer architecture education, instruction level parallelism, simulators, superscalar microprocessors, Tomasulo algorithm, Scoreboard algorithm

1 Introduction

A course in advanced Instruction Level Parallelism (ILP) covers both basic hardware options for ILP extraction: VLIW and Superscalar architectures. While the former one uses the compiler to extract the ILP, the later is based on hardware scheduling. The generic topics covered include (1) basics of ILP, (2) out-of-order execution, (3) speculative execution, (4) algorithms for ILP extraction, (5) superscalar and VLIW techniques [1, 2, 3, 4].

Since VLIW and its advanced successor EPIC (Explicit Parallel Instruction Computer) rely on software, the best way to teach them is to introduce an actual compiler for the architecture. We are in the process of realizing it for the first time using an Itanium based system. On the other hand, the superscalar execution focuses on the hardware elements that support out-of-order execution and

Manuscript received February 2, 2004
The author are with Faculty of Natural Sciences and Mathematics, Arhimedova 5, 1000 Skopje, University of Ss. Cyril and Methodius, Skopje, Macedonia, E-mail: [anastas,marjan]@ii.edu.mk

speculative execution using branch prediction. Using architecture visualization tools [5, 6] enables the students to comprehend the data and control flow, which again eventually should facilitate them to produce more efficient code. The website *http://twins.ii.edu.mk* hosts details of implementation and lab exercises for these simulators. In this paper we present SuperSim simulator that is powerful tool for visual simulation of internal organization of dynamic out-of-order ILP processor. Another simulator is used for simulating the Tomasulo approach for organization of register renaming techniques with reservation stations. The last simulator shows implementation of Scoreboard approach for reservation stations. At the end, we give an overview of the results achieved by the students in the last two generations that we use the simulators.

2 Description of SuperSim simulator

ILP concepts and topics were covered in the Microprocessors course in previous years. In this course the students had only chance to work with the simulator without any homework assignments or projects.

Starting from 2002/2003 a new ILP course is set in the Computer Science studies. In the ILP course curricula the students are assigned with two projects. The second project covers several aspects of the dynamic superscalar execution. It was performed using our own superscalar simulation platform SuperSim [5, 6]. The main features of the SuperSim Simulator are:

- Running user code, written in its own pseudo assembler

- Syntax checking of the user code with error indication

- Extensive configuration

- Simulating a big range of processors, varying from simple RISC to advanced PostRISC, 1.

- Step by step execution

- Visual representation of each stage of the pipeline

- Fast, non visual mode for better performance

- Vast logging capabilities for performance analysis

- Detailed statistics

The internal architecture of simulated processors (1) consists of instruction cache, instruction issue window, ROB (reorder buffer), register file and rename buffer, reservation stations, execution units and data cache.

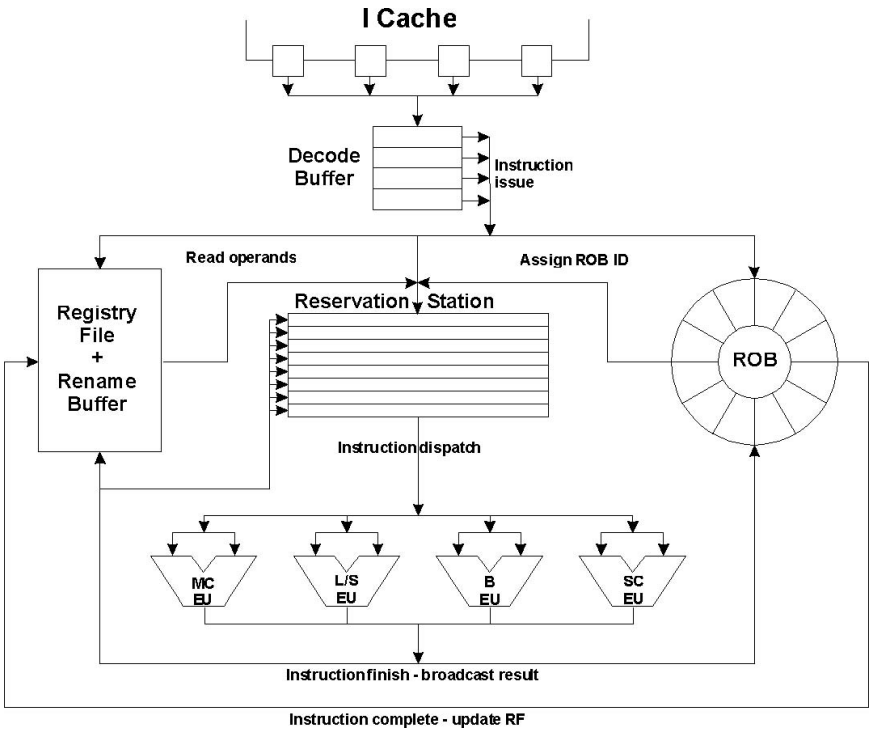

Fig. 1. Architecture of the simulated processors

3 User Interface of SuperSim simulator

The simulator has a very friendly user interface. It consists of several separate windows, including code editor, runtime, configuration, statistics and other windows.

The code editor window (2) enables the user to write its own custom code, using the pseudo assembler. The code can be saved into a file or loaded from one. Options available on this window include syntax checking with indication of possible errors and standard file management. Code can have inline comments, separated with '//' from the instructions. Especially important is the configuration option, which defines the simulated execution environment.

The configuration window (*Fig III*) consists of several major parts, each represented with a special tab. The configuration enables choosing the number and the type of the execution units. The maximum number of execution units is 6, and the minimum is 1. Supported units are

1 multi cycle unit, for execution of multi cycle integer operations, like division or multiplication

Up to 3 single cycle integer units, for execution of simple integer arithmetic

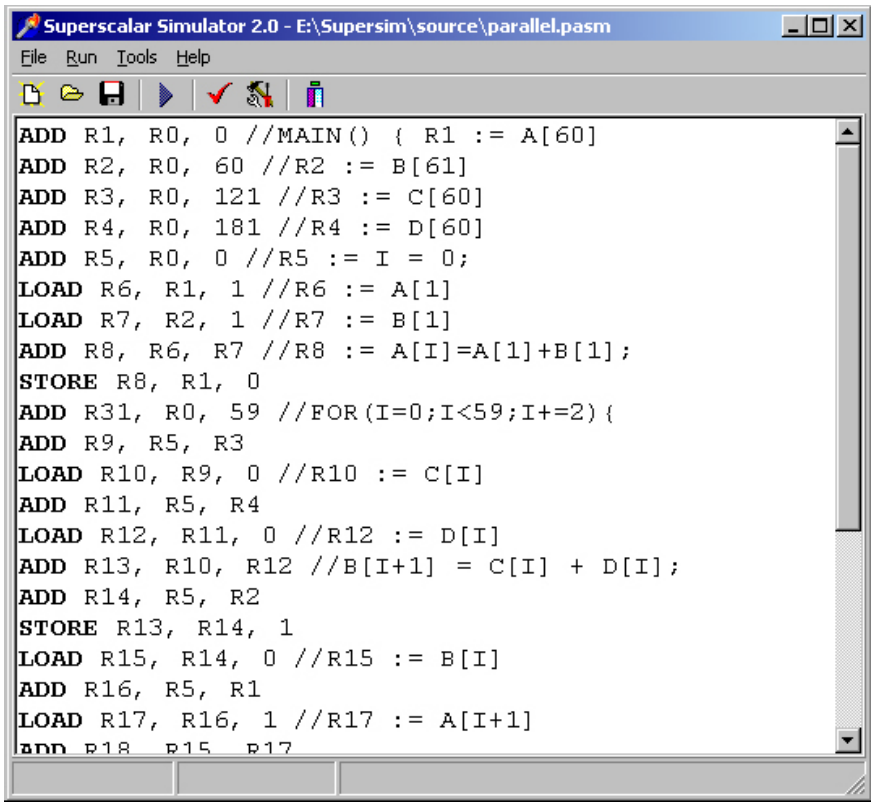

Fig. 2. Code editor window

1 load/store unit for address calculation of the memory transfer instructions and 1 branch unit for calculation of the branch target addresses.

Only the multi cycle unit is mandatory, while the others can be added or removed. If a special unit is not used, for example the load/store unit, the multi cycle unit performs the operations. The issue rate can also be configured on this tab, varying from 1 up to the total number of units used.

The second tab of the configuration window covers the use of shelving. When shelving is used, the user can select between central or dedicated reservation stations. For each station used, the number of entries can also be configured.

The next tab is used for configuring the register renaming options of the simulator. If renaming is used, the number of rename buffers can be selected. Additionally, the access method for the renamed registers can be chosen from indexed or associative.

The ”Out of order” tab, enables the using of the out of order issue and dispatch. On the same tab, the user can adjust the number of entries in the Reorder Buffer

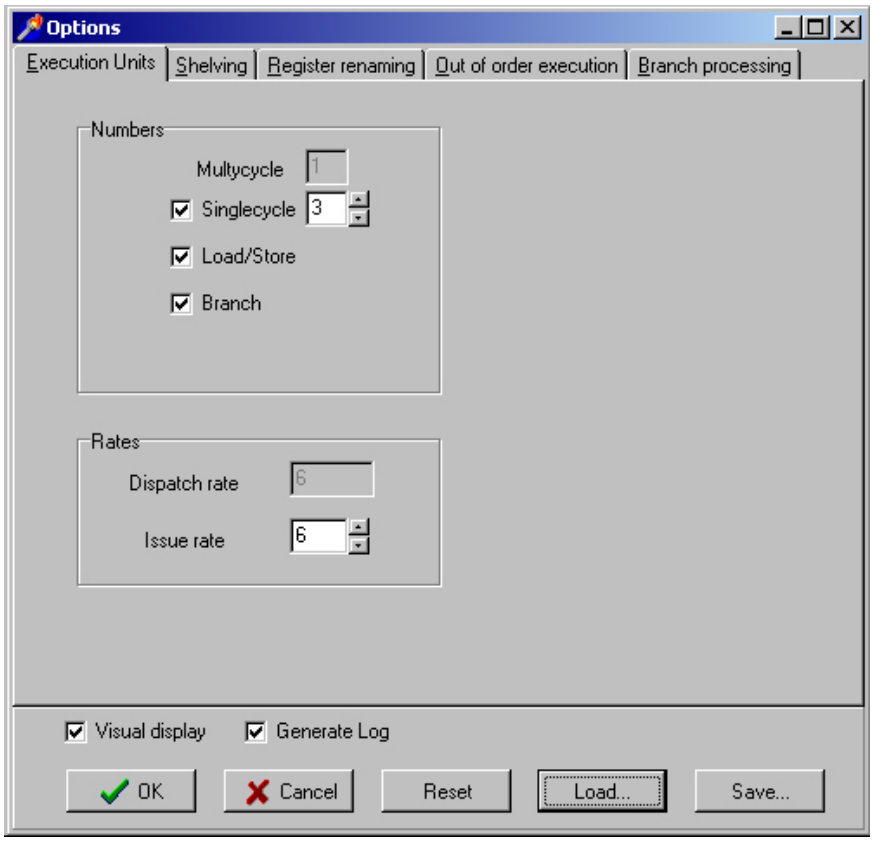

Fig. 3. Configuration window

(ROB).

The final configuration tab covers the branch processing used in the simulation. It can be blocking or speculative. When using speculative branch prediction, three modes are available: fixed, static and dynamic. The dynamic branch processing can be configured to use BTAC, BHT or both. It can also use global 2-bit history, for better prediction.

Other available options are turning on and off the visual simulation, which can increase performance and tuning on and off the logging option. When visualization is disabled, the number of clock cycles simulated per second improves by 7-10 times.

The selected configuration can be saved into a file for later reuse, or loaded from one.

4 Runtime SuperSim Simulator Environment

The runtime environment greatly depends on selected configuration. When all the options are enabled, it looks like 3. The top part consists of some command buttons, among which are:

"Close" for closing the runtime window

"Run" for running the simulation continuously

"Step" for executing cycle by cycle

"Pause" for pausing the simulation when ran in continuous mode

Depending on the configurations some or all of the buttons in the upper right part will be enabled:

"Show ROB" displays the ROB,

"Show RF" displays the registry and rename registry file,

"Show BT" displays the branch prediction tables window,

"Show DC" displays the data cache,

The rest of the window is divided into separate parts for each stage of the pipeline. Mandatory stages are Fetch, Issue, Execute and Write-back, while the other two, Dispatch and Complete are shown only if shelving and out-of-order execution are used, respectively. For each stage, a container represents the appropriate tables and/or buffers that hold the current instructions. In the upper left part, two separate containers represent the pending load and store queues.

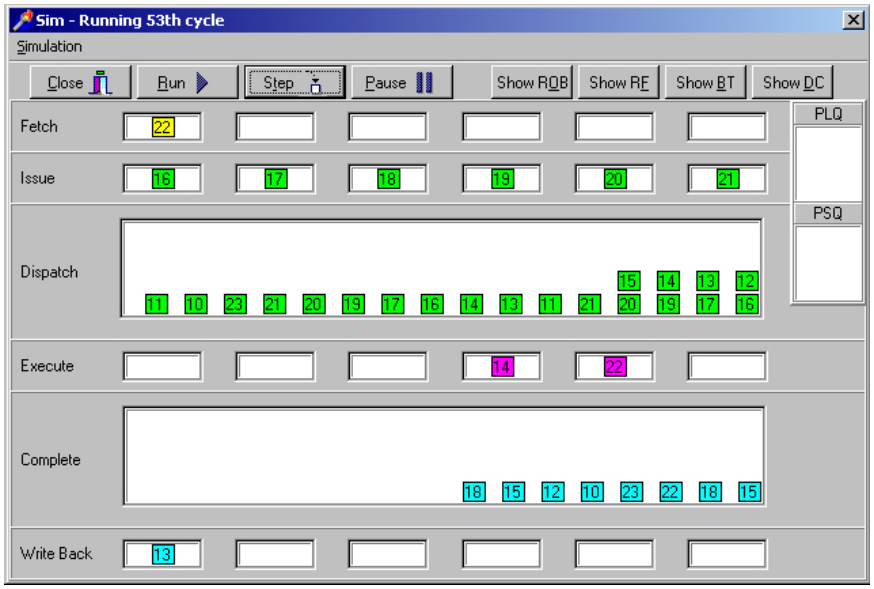

Fig. 4. The runtime interface of the simulator

The ROB window is used for monitoring the work of the reorder buffer. It has an entry for each instruction that has been issued and has not completed yet. Since the ROB is designed as a circular buffer, at also shows the head and the tail pointer in the buffer. Instructions are represented in different colors, depending on the stage of the pipeline they are in.

The registry file window shows the state of both the architectural and the rename registers. On the left of the window, architectural registers are shown. For each rename register, there are three parameters shown: the number of the architectural register that is mapped to this rename register, the value (if calculated yet) and the latest bit.

The branch tables' window is used for monitoring the state of the branch prediction tables. Depending on the configuration, one or two tables are shown. They are the BHT and/or the BTAC.

The data cache window shows a map of the data memory, with each entry representing a 4-byte word.

The statistics window, gives a detailed statistics of the simulated code and configuration. The figures include the total number of executed instruction of each type, branch statistics and prediction accuracy measures, the flow of the instruction through each stage and both memory and register data dependencies. Some advanced measures are also included, like the average number of cycles required for flushing the processor and average number of register wasted when a miss-prediction occurred.

The instruction set of the simulator represents a subset of the standard modern instruction sets, and contains the instructions shown in 4.

The simulator simulates a processor performing 32-bit integer operations. The floating-point part is not considered in this project. Most of the current PostRISC features can be simulated using the SuperSim, including out-of-order issue, register renaming, shelving, branch prediction etc.

Supported memory addressing modes are displacement and indexed based. While the same mnemonic is used for both modes, instruction processing is different depending on the mode. The memory is divided into instruction cache and 1024 locations of 32-bit words data cache. The memory is aligned on a word (4 bytes) boundary and all memory access instructions refer to a word address.

The maximum number of execution units is six. Instructions that take multiple clock cycles to execute, i.e. the 'mul' instruction, are executed in the multi-cycle, which is obligatory. Optionally there can be up to three single-cycle execution units for instructions like 'add', or 'sub' that take one clock cycle to execute, one load/store unit for handling memory access, and one branch unit dedicated for branch processing. When there is no available corresponding execution unit, the instructions are executed in the multi-cycle unit, which provides the functionality

Instruction	**Semantics**	**Comment**
ADD R1, R2, R3	Regs[1] = Regs[2] + Regs[3]	
SUB R1, R2, R3	Regs[1] = Regs[2] - Regs[3]	
AND R1, R2, R3	Regs[1] = Regs[2] & Regs[3]	
OR R1, R2, R3	Regs[1] = Regs[2] | Regs[3]	
NOT R1, R2, RX	Regs[1] = ! Regs[2]	The third operand can be
SHL R1, R2, R3	Regs[1] = Regs[2] SHL Regs[3]	either register,
SHR R1, R2, R3	Regs[1] = Regs[2] SHR Regs[3]	or a constant
MOD R1, R2, R3	Regs[1] = Regs[2] Modulo Regs[3]	
DIV R1, R2, R3	Regs[1] = Regs[2] / Regs[3]	
MUL R1, R2, R3	Regs[1] = Regs[2] * Regs[3]	
LOAD R1, R2, 200	Regs[1] = Mem[Regs[2] + 200]	Reads a word from memory
STORE R1, R2, 150	Mem[Regs[2] + 150] = Regs[1]	Writes a word in memory
BEQ R1, R2, 200	if (Regs[1]=Regs[2]) IP = IP+200	
BNE R1, R2, R3	if (Regs[1]!=Regs[2]) IP = IP+Regs[3]	The third operand can be
BGT R1, R2, 200	if (Regs[1]¿Regs[2]) IP = IP+200	either register,
BLT R1, R2, R3	if (Regs[1]¡Regs[2]) IP = IP+Regs[3]	or a constant
BGE R1, R2, 13	if (Regs[1]¿=Regs[2]) IP = IP+13	
BLE R1, R2, R3	if (Regs[1]¡=Regs[2]) IP = IP+Regs[3]	

Table 1. : Instruction set

of all execution units. The number of execution units determines the dispatch rate so there are no restrictions about the instructions being dispatched. Issue rate can be set up to the dispatch rate.

The use of RS is optional. When selected, there is a choice between central or dedicated RS. Dedicated RS are placed in front of every execution unit, so the issue stage directs every instruction to the corresponding RS. In the case of central RS there must be additional logic to determine the execution unit where the instruction is dispatched. Additional requirement in the case of central RS is the number of output and input ports, which have to be larger unlike the case of dedicated RS.

Register renaming is implemented by separate register rename file (also known as rename buffer). The access to the rename buffer can be associative or indexed.

When using associative access, there may be multiple instances of renames of one architectural register with separate notion of the last rename. In contrast only one rename per architectural register may exist with indexed access.

Out of order execution refers to whether instructions are issued out of order or dispatched out of order. When shelving is enabled instruction issuing is in order, while instruction dispatch is out of order. This design option is realized since the issue stage does not check for dependencies so there cannot be pipeline blockages due to dependency. If shelving is disabled, the only possibility is out of order instruction issue.

If branch processing is speculative, predictions about branch instructions can be: fixed ”always not taken”, static displacement based, or dynamic with optional use of BTAC, BHT or 2 bit global history register. In the latest case BTAC is used only for recent taken branches and the use of either BTAC or BHT is obligatory if dynamic prediction is selected. Additionally, when BHT is used, global BHT can be activated and the initial state can be set.

The SuperSim simulator is developed using Borland Delphi and targets 32-bit Windows platforms. It has full object oriented design, with each phase in the pipeline represented by its own object. Each object has a public interface for realization of communications between the stages in the pipeline. The object architecture makes upgrading easy and intuitive.

The performance in the sense of simulated clock cycles per second varies depending on whether the visualization is on or off. When off, it simulates around 100 clock cycles per second, measured on PIII working on 650MHz. If visualization is on, this number is 7-10 times smaller.

5 Lab practising with SuperSim simulator

The simulator was used as a main tool for realization of the laboratory exercises. After getting to know the tool, each student had to develop and analyze a specific sample program.

The programs simulated included searching, sorting, prime number search, SCD, matrix operations, linked list operation, conversions etc. The analysis concerned the performance impact of the key ILP factors like the number of execution units, number of register available for renaming, type of the reservation stations, ROB entries, loop unrolling and branch prediction techniques. The deliverables were the program itself and a paper explaining the results of the analysis.

For example, a student had to write a simple binary search program and analyze the influence of the number of the execution units, number of registers available for renaming and type and size of the reservation stations.

ADD R1, R0, 1	STORE R7, R0, 14
ADD R2, R0, 2	STORE R8, R0, 15
ADD R3, R0, 3	STORE R9, R0, 16
ADD R4, R0, 4	STORE R10, R0, 17
ADD R5, R0, 5	ADD R1, R0, 10
ADD R6, R0, 6	ADD R2, R0, 7
ADD R7, R0, 7	ADD R3, R0, 0
ADD R8, R0, 8	ADD R4, R1, -1
ADD R9, R0, 9	ADD R5, R3, R4
ADD R10, R0, 10	DIV R5, R5, 2
STORE R1, R0, 8	LOAD R6, R5, 8
STORE R2, R0, 9	BGT R2, R6, 3
STORE R3, R0, 10	ADD R4, R5, 0
STORE R4, R0, 11	BLT R2, R6, 2
STORE R5, R0, 12	ADD R3, R5, 0
STORE R6, R0, 13	BEQ R6, R2, 2
	BNE R6, R2, -8

Table 2. Sample program

The pseudo assembler code of the sample program is given in 1.Some of the student's conclusions are presented in 2 and 3.

Exec. Units	Issue rate	Access	Ren Regs	Cycles
2	2	associative	1	70
2	2	associative	2	53
2	2	associative	3	49
2	2	associative	4	47
2	2	associative	5	47
2	2	associative	6	47
2	2	associative	32	47
2	2	indexed	32	49

Table 3. : Impact of the register renaming

6 Simulator of Tomasulo Algorithm

Another very useful tool used in the teaching of the course is a simulator of the Tomasulo algorithm. The simulator features configurable execution core, variable issue rate and variable latency per instruction class.

Exec. Units	Issue rate	Type	Entries	Cycles
2	2	central	1	97
2	2	central	2	67
2	2	central	3	61
2	2	central	4	60
2	2	central	8	60
2	2	central	20	60
2	2	individual	20	55

Table 4. : Impact of shelving

The program segment to be simulated can be changed by adding and deleting the last instruction of the segment. For each instruction, the destination and the source registers can be specified. The supported instructions are ADD, SUB, MUL, DIV, LOAD and STORE. Each type of instruction can have its own latency, ranging from 1 to 20. The ADD and SUB instruction are executed in the simple integer EUs, while MUL and DIV are executed in the multi-cycle EUs. There are also dedicated units for the execution of the memory transfer. The number of units is also configurable, and can be set to 2 or 3 units of each class.

The simulator has a very rich visual interface, as shown in 4. It illustrates the movement of the instructions to the reservation stations, RS entry status and register occupancy. This simulator is fully Java realized and operates in Internet environment from anywhere on all possible platforms and operating systems. There is a very interesting approach for user data entry. Each instruction is added to program by clicking on several possible options, giving the freedom to choose and restricted to only possible instruction set.

The simulation can be operated in all at once mode or in a step by step mode. Usual interface allows a step forward and all changes, data dependencies and data flow are shown. A very interesting and helpful feature is the step backward, allowing better understanding of the algorithm. This is very unusual feature, but we found very interesting and helpful in this simulator. We also advise future developers to implement such a feature since it gives a freedom for research and deep analysis, without restarting paradigm known in computer systems.

The simulator helped the students to better recognize the process of register renaming. By executing different examples of code, the students could compare the effects of the algorithm on the performance.

Prior to introducing the simulator, the students were given homework regarding the Tomasulo algorithm. Their task was to simulate by hand a short sequence of instructions, filling the appropriate tables for each step of the execution. They had to fill the status table of the reservation units and the register status table. Also, they

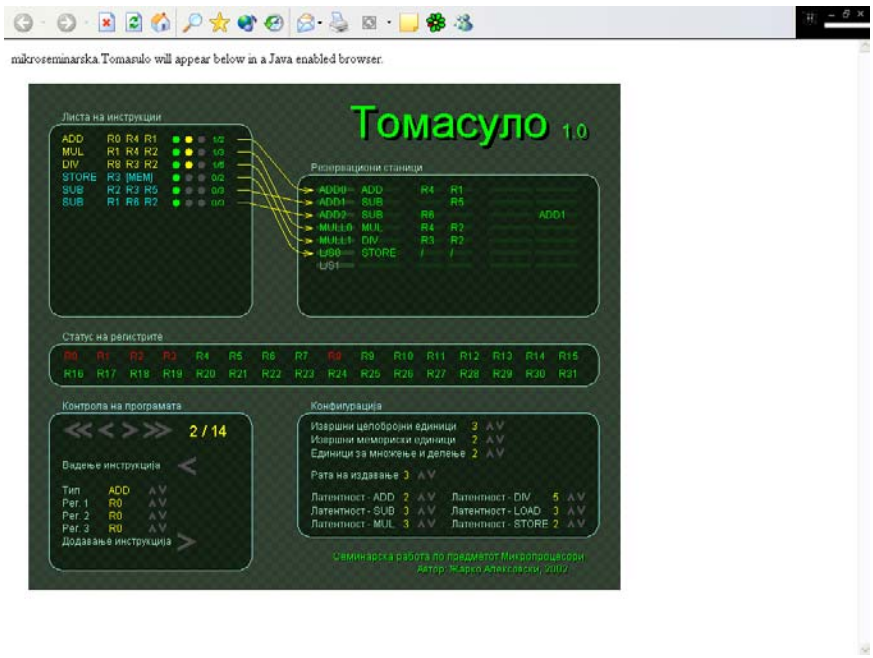

Fig. 5. The interface of the Tomasulo algorithm simulator

were posed some questions regarding the status of the Tomasulo tables at a given cycle of the execution of some short instruction segment.

Using the simulator, the students were assigned with homework to realize step by step execution of simple programs. They had to realize all data dependencies and realize how they are resolved in ILP organization with several reservation stations. It is a very interesting approach to realize deadlocks, and delays due to conflicts. The most important gain is that instead of the boredom filling of tables, now the students can focus on tuning the various parameters to get better results.

7 Scoreboard simulator

Yet another simulator has been developed to provide better understanding of the scoreboard algorithm. The simulator offers a complete environment, including program editor, file management, configuration and run-time interface. It is build with Borland Delphi and works on Windows operating systems environment.

The simulator uses single file to store a complete environment, which includes the code to be simulated along with the options configured.

To use this program you need to follow these steps:

Load or Create a program segment i.e. instructions.

Select an appropriate configuration (if not it will use default settings).

Start the simulation, using the control buttons.

The syntax and the instructions that the program can recognize are the following:

[Instruction] [Operand1] [Operand2] [Operand3]

Instructions can be: ADD, SUB, MUL, DIV, LOAD, STORE and operands can be: R0, R1, R2, . . . R31.

Using the configuration window, students can specify the issue rate and the number of various execution units and their latencies. Also, the simulation speed can be configured to get a better view of the simulation flow, as shown on Fig VII.

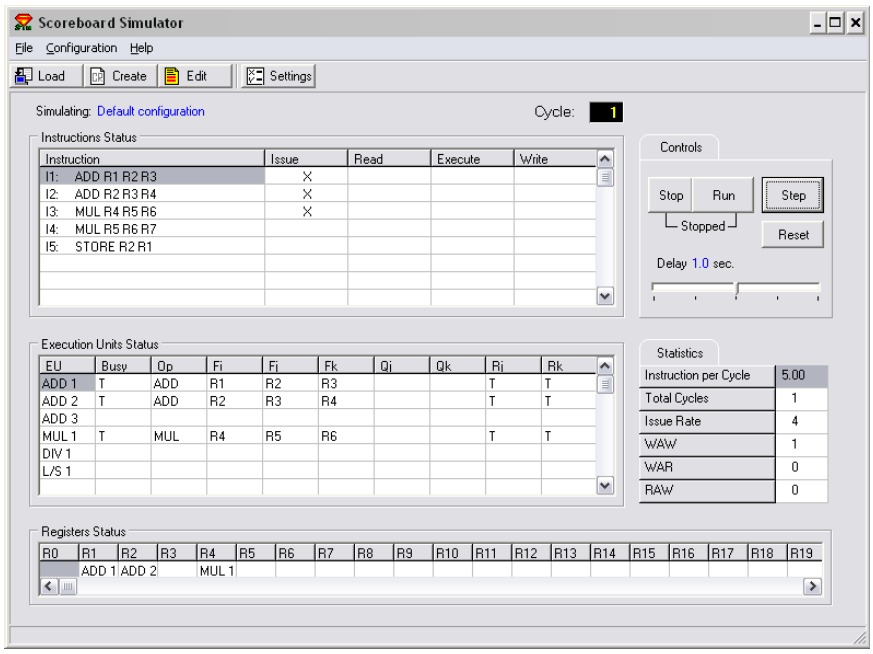

Fig. 6. The interface of the Scoreboard simulator

The runtime window includes the status of the algorithm tables, status of the execution units and registers. It also offers dynamic statistics related to the IPC, total cycles executed and the frequency of the data dependencies.

Before the simulator was built, the students were given hand-written homework about the Scoreboard algorithm. They had to simulate the step-by-step execution of some short sequence of instruction and for each cycle, fill in the appropriate tables. The tables included the status of the registers, execution units and instructions.

After the scoreboard simulator was introduced in the teaching process, the students had to perform the same analysis using it. They had the chance to see if they

did the manual simulation well, compare the results and learn from it. Also, they had the chance to explore the effects of the algorithm on larger examples, ones that were too much to be done manually. They also had to see for themselves the impact of the algorithm on the overall processor performance.

8 Achieved Results

51 students enrolled in the advanced ILP course "Advanced Computer Systems" for first time held at Computer Science Studies. The course was finished successfully by 76% of the students by the end of the semester. A surprising fact is that the course average is 87%, which is very high compared to other courses and average scores. The course statistics are presented in [7] with correlation of homework assignments and projects.

The students complain to the timing requirements, they had more time to complete the first project then the second and we are going to improve it in future. Similar advanced ILP course existed in the old Computer Science Studies curriculum in the "Microprocessors and microcomputers" course. It covered ILP concepts and topics but lacked VLIW organization and compiler techniques. All the students had to make only one project given at the end of the course, without time limit. However, there were no deadlines, so it took the students almost a year to complete the project, i.e. they finished the project before taking the exam. This was the reason to set a real deadline for the project.

Average score on learned ILP concepts in previous years was lower then this year. Previous years we used simulators in courses only during the lectures for presentation purposes. We realize that improved achievement is due to the extensive use of simulators on lab practicing, homework assignments and projects. Not only they learned ILP concepts, but however, they get deeply into ILP processor architecture realizing how to program modern processors and exploit maximum parallelism and performance.

9 Conclusion

The new ILP course started 2002/2003. We introduced a lot of innovation in this course concerning grading scale and evaluating students' activities, homework assignments and projects [7]. For first time we setup a special project and homework assignments about ILP simulators, instead of just visual presentation during lectures.

Students found homework assignments and project realized with ILP simulators a very interesting tool to obtain knowledge and specially to make further re-

search on ILP processor behavior. It helped them not just learn main concepts and topics, but deeply get into computer architecture and analyze reasons for deadlocks and stalls due to data dependencies in conditions of high parallelism on instruction level. They widely accepted challenges for ILP processor architecture and exploit maximum performance.

This innovative approach showed not just greater students' interest, but also approved with greater students' achievement and average score. The number of students that passed the exam at the end of course is also greater then average, or compared to corresponding number of students for the course with similar topics in previous Computer Science Studies curricula.

The usage of the simulators continued with the next generation of students. As an indication, out of 66 students, 61 passed the exam through colloquia, homework and project assignments, with an average grade of 8,4. The distribution is given in Fig. VIII.

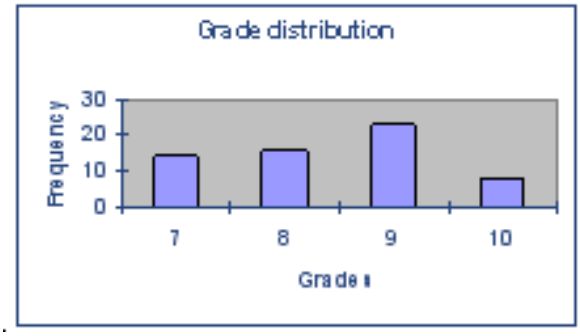

Fig. 7. The distribution of grades

Compared to the previous experience and to other courses, the results are more than satisfactory, proving the efficiency of the simulators in the teaching process.

References

[1] J. L. Hennessy and D. A. Patterson , *Computer Organization and Design, the hardware/software interface*, 2nd ed. San Francisco, California: Morgan Kaufmann Publishers Inc., 1998.

[2] D. A. Patterson and J. L. Hennessy , *Computer Architecture: A Quantitative Approach*, 3rd ed. San Francisco, California: Morgan Kaufmann Publishers Inc., 2003.

[3] M. Gusev, *Contemporary Computer Systems*. Skopje: Medis Informatica, 1998.

[4] ——, *ILP Instruction Level of Parallelism*. Skopje: Faculty Natural Sciences and Mathematics, 2001, internal textbook.

[5] M. Gusev, A. Misev, and G. Popovski, “Simulation of superscalar processor,” in *proc. of ITI’98*, Pula, Croatia, 1998, pp. 169–174.

[6] A. Misev and M. Gusev, “Supersim v2.0 ilp processor visual simulator,” in *Computation Intelligence and Information Technologies, Proceedings*, R. Stanković, Ed. Niš, Yugoslavia: Faculty of Electonic engineering, June 20-21, 2001, pp. 161–166.

[7] M. Gusev, “New methodology and evaluation system,” in *Proc. TEMPUS CD JEP 16160-2001 project workshops*, 2003.