Visual simulator for ILP dynamic OOO processor

Anastas Misev, Marjan Gusev
Institute of Informatics, PMF, Sts. Cyril and Methodius University
Arhimedova 5, PO Box 162, 1000 Skopje, Macedonia
anastas@ii.edu.mk, marjan@ii.edu.mk

Abstract

The purpose of this article is to provide an introduction to the SuperSim simulator for ILP processors as a teaching tool for computer architecture related courses. It presents the various aspects of the simulator, including the user interface, the instruction set, the configuration possibilities and applications. The main focus is on the educational usage of the simulator, through the experience gained in its actual application.

1. Introduction

Superscalar processors are one of the two major directions of ILP development. They issue multiple instructions per cycle, which results in complex decoding stage. This can lengthen the clock cycle or lead to multiple decoding cycles. Usually superscalar processors employ some kind of predecoding of instructions while they are fetched from memory to instruction cache. Pre-decode bits are attached to every instruction usually indicating the instruction class and the type of required resources.

Another aspect of multiple instruction issue is that can lead to higher performance, but at the same time it *amplifies* the restrictive effects of control and data dependencies on the processor performance. In order to reduce these effects, superscalar processors employ advanced techniques like register renaming, shelving and speculative branch processing.

Developing powerful microprocessors requires research in many different areas; such are electronics, algorithms, optimization, etc. Many new techniques are required for this process. To prove their efficiency, in a manner that allows grater freedom of research, simulation tools are very important.

The usage of simulators in the computer architecture courses has been proven as the best approach towards students' better understanding of the main architectural concepts. This is especially true for the visual simulators, since many internal features can be best understood through dataflow visualization.

2. Description of the SuperSim Simulator V 2.0

The basic considerations for designing the SuperSim Simulator were taken from the design space concept given by Sima et al [12], using similar experience of [2]. The previous versions of the simulator are covered in [7].

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
- 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

3. User Interface

The simulator has a very friendly user interface. It consists of several separate windows, including the code editor (Fig.1), runtime, configuration, statistics and other windows.

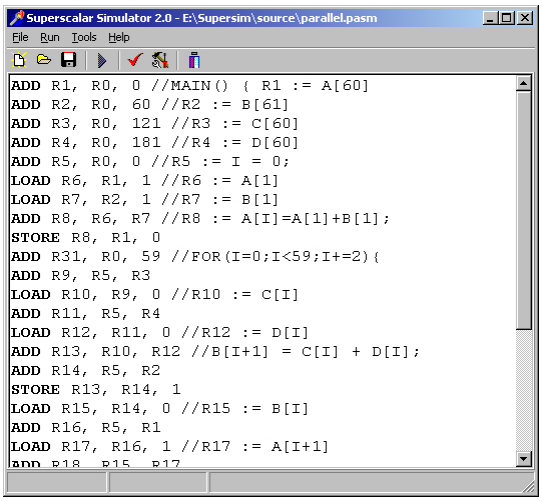

Figure 1: The code entry window

The code editor window 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.

4. Configuration

The configuration window consists of several major parts, each represented with a tab, as shown in fig. 2. 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
- 1 load/store unit for address calculation of the memory transfer instructions and
- 1 branch unit for calculation of the branch target addresses.

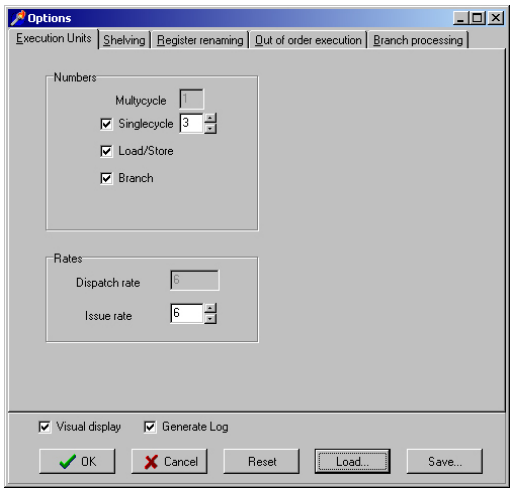

Figure 2: The options window

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, shown in fig. 3, 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, fig. 4, 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.

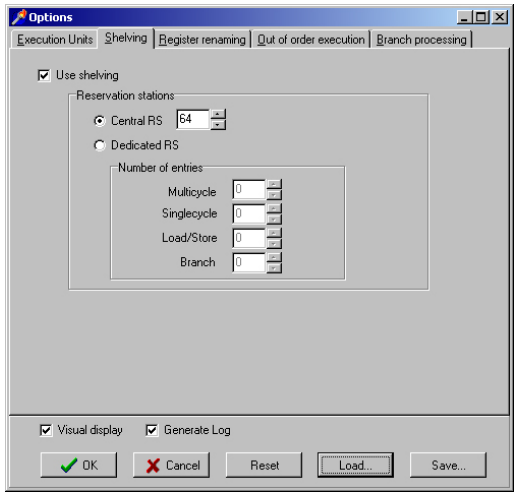

Figure 3: Shelving options

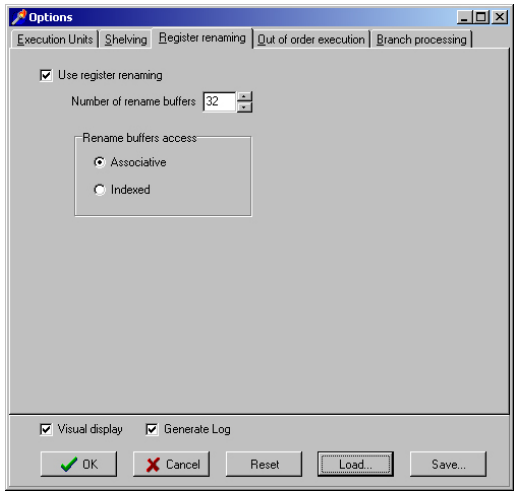

Figure 4: Register renaming options

The "Out of order" tab, fig. 5, 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 (ROB).

The final configuration tab covers the branch processing used in the simulation, as shown in fig. 6. 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 options available 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 is 7-10 times bigger.

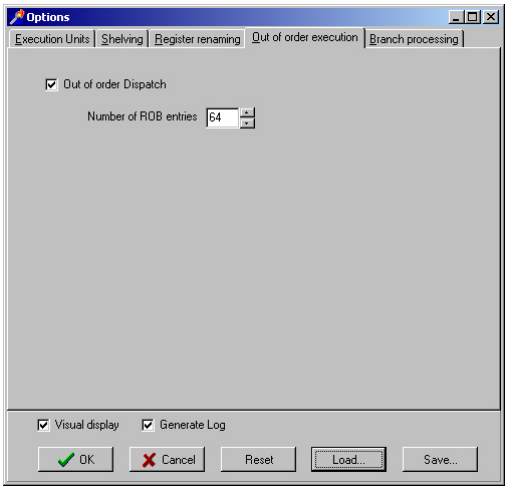

Figure 5: Out-of-order options

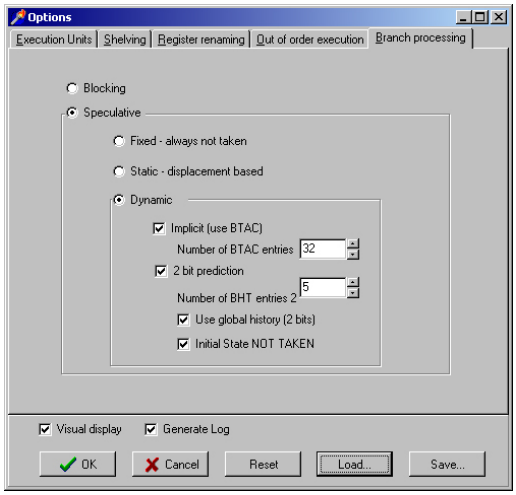

Figure 6: Branch processing options

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

5. Runtime

The runtime environment greatly depends on the selected configuration. When full configuration is used, it looks like in fig. 7. 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, fig. 8, "Show RF" displays the registry and rename registry file, fig. 9, "Show BT" displays the branch prediction tables window, fig 10, "Show DC" displays the data cache, fig. 11.

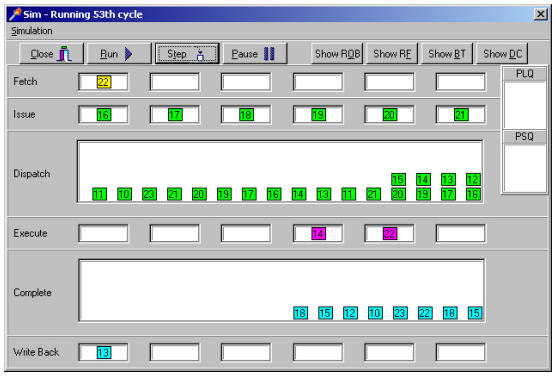

Figure 7: The runtime window

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.

The ROB window, shown in fig. 8 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.

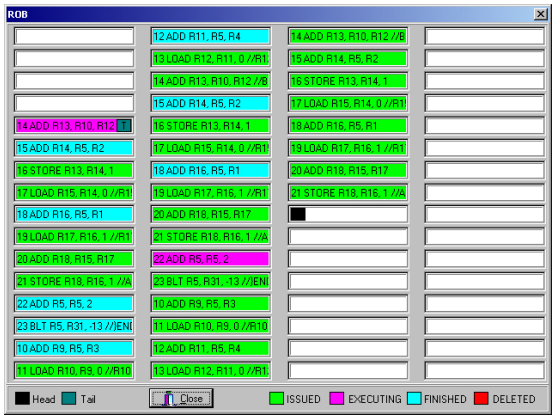

Figure 8: The ROB window

The registry file window, fig. 9, 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.

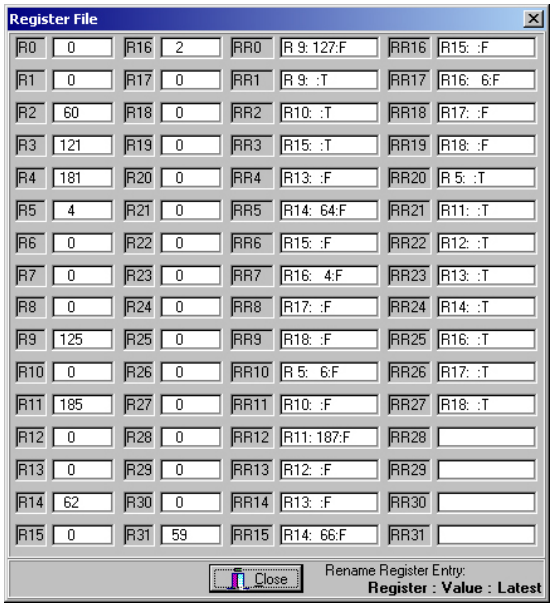

Figure 9: The Register file window

The branch tables' window, fig. 10, 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.

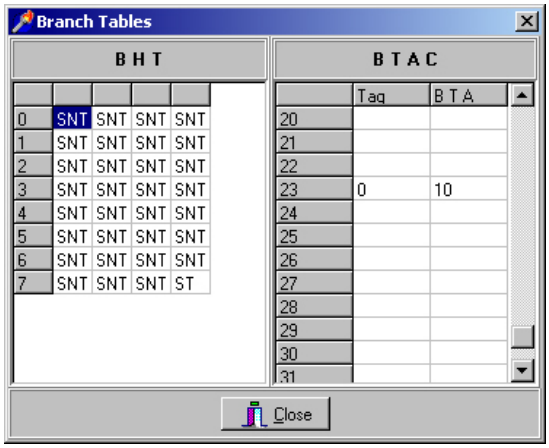

Figure 10: The Branch tables' window

The data cache window shows a map of the data memory, with each entry representing a 4-byte word, as shown in fig.11.

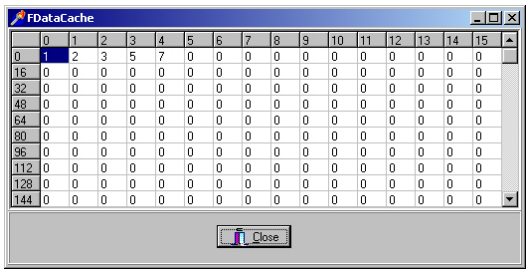

Figure 11: The Data cache window

The statistics window, shown in fig. 12, 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.

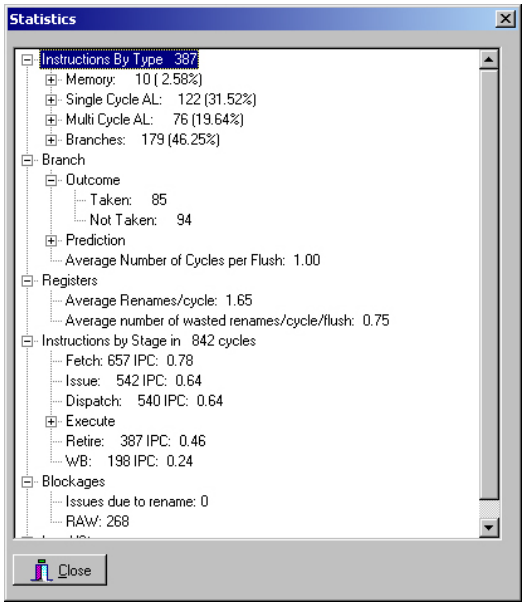

Figure 12: The Statistics window

6. Internal design

The instruction set of the simulator represents a subset of the standard modern instruction sets [6,9,11], and contains the instructions shown in table 1.

The simulator simulates a processor performing 32-bit integer operations with block diagram presented in fig.13. The floating-point part is not considered in this project. Most of the current PostRISC features [6, 9, 11] 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 [6]. 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 (refer to fig. 2). 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

Table 1: Instruction set

Instruction	Semantics	Comment
ADD R1, R2, R3	Regs[1] = Regs[2] + Regs[3]	The third operand can be either register, or a constant
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]	
SHL R1, R2, R3	Regs[1] = Regs[2] SHL Regs[3]	
SHR R1, R2, R3	Regs[1] = Regs[2] SHR Regs[3]	
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	The third operand can be either register, or a constant
BNE R1, R2, R3	if (Regs[1]!=Regs[2]) IP = IP+Regs[3]	
BGT R1, R2, 200	if (Regs[1]>Regs[2]) IP = IP+200	
BLT R1, R2, R3	if (Regs[1]<Regs[2]) IP = IP+Regs[3]	
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]	

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 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 [12].

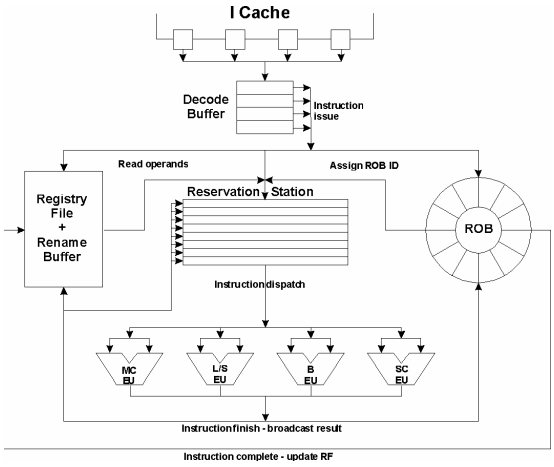

Figure 13: Block diagram of the simulator

The use of RS is optional with the possibilities shown in Fig.3. 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) [1,3,5,12,13,14,15,16]. 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. Fig.5 shows the possible options about out of order execution [15,16].

Branch processing options are shown in Fig.6. 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 [17]. Additionally, when BHT is used, global BHT can be activated and the initial state can be set.

7. Implementation

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.

8. Teaching ILP using the simulator

The SuperSim simulator can be equally well used in research and in education. Its visual interface helps students to understand the functionality of a RISC or PostRISC, get familiar with the basic concepts of ILP and practice their assembly language programming skills.

The simulator executables, with sample configurations and programs are available to the students through the computer architecture courses web sites. After the initial introduction of the basic simulator elements and performing some simple examples, each student is assigned a project. The project consists of writing a small assembly program (searching, sorting, prime number search, SCD, matrix operations, linked list operation, conversions etc.) and performing some analysis of the superscalar techniques on the program execution. 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.

The results of this method of teaching ILP were more than satisfactory. The students' interest for the course was bigger and the achieved results were better then before the introduction of the simulator [10].

9. References

[1] Austin,T., Sohi,G.S., (1992), *Dynamic Dependency Analysis of Ordinary Programs,* Proc.19th Int.Conf. on Computer Architecture, ISCA-19.

[2] Burger, D., Austin, T., (1997), *The SimpleScalar Tool Set, Version 2*, Technical report of the University of Wisconsin-Madison, Computer Science Department.

[3] Conte T.M. (1996), *Superscalar and VLIW processors*, in *Parallel and Distributed Computing Handbook*, ed. by A.Y.Zomaya, McGraw Hill.

[4] Farcas, K. et all, (1995) *Register File Design Considerations in Dynamically Scheduled Processor,* WRL Technical report 95/10, Paolo Alto, California.

[5] Gonzalez, J. and Gonzalez, A., (1995) *Identifying Contributing Factors to ILP*, Univesitat Politecnica de Catalunya, Barcelona, Spain.

[6] Gusev M. (1998), *Contemporary Computer Systems,* Medis, Skopje, Macedonia.

[7] Gusev, M., Misev, A. and Popovski, G. (1998) *Simulation of Superscalar Processor,* proc. of ITI'98, Pula, Croatia.

[8] Gusev,M, Misev.A and Popovski.G (1999), *Memory Address Dependencies,* ITI-99 pp.191-196.

[9] Hennessy J.L., Patterson D.A. (1998), *Computer Organization and Design: The Hardware Software Interface*, sec. edition, Morgan Kaufmann Publishers, San Francisco, California.

[10] Misev, A., Gusev, M., (2004), *Simulators for ILP courses*, proc. of TEMPUS CD JEP 16160 workshops, Nish, Serbia and Montenegro.

[11] Patterson, D and Hennessy, J. (1996) *Computer Architecture A Quantitative Approach,* second edition, MKP, San Francisco, California.

[12] Sima D. at al. (1997), *Advanced Computer Architectures: A Design Space Approach*, Addison Wesley Longman, Harlow, England.

[13] Smith, J. and Pleszkun, A. (1988) *Implementing Precise Interrupts in Pipelined Processors,* IEEE Transactions on computers, vol. 37, no. 5, p.562-573.

[14] Tyson,G., Austin,T. (1997), *Improving the Accuracy and Performance of Memory Communication Through Renaming*, Proc.30th Ann. Int. Symp. on Microarchitecture, MICRO-30.

[15] Wall, D. (1993) *Limits of Instruction-Level Parallelism,* WRL Research report 93/6, Paolo Alto, California.

[16] Wall, D. (1994) *Speculative Execution and Instruction-Level Parallelism,* WRL Technical Note 94/42, Paolo Alto, California.

[17] Yeh, T.Y., Patt, Y. N., (1992) *Alternative Implementations of two-level Adaptive Branch Prediction*, Proceedings of 19th Int. Symposium on Computer Architecture (ISCA) pp.124-134.