SIMIPS : a cycle-precise interactive simulator for teaching microprocessor architecture

Y. Body, F. Dromard, A. Greiner, M-M. Paget, F. Pétrot

Université P. et M. Curie, Laboratoire LIP6, Equipe ASIM
4 Place Jussieu, F75252 Paris Cedex 05
FRANCE

E-mail : Yves.Body@lip6.fr
Francois.Dromard@lip6.fr
Alain.Greiner@lip6.fr
Marie-Martine.Paget@lip6.fr
Frederic.Petrot@lip6.fr

http://www.asim.lip6.fr

Abstract

We have developed a simulation environment for a microprogrammed version of the MIPS R3000 microprocessor. This education tool shows the execution mechanism of instructions on the hardware structure at register transfer level. The execution of each instruction can be achieved in one simulation step or split up into a sequence of micro-instructions, each transfert being then executed in one machine cycle.

Keywords : simulation, interactive learning, MIPS R3000 processor, microprogramming

1. Introduction

Simulation, focus of a tremendous amount of research, is used since many years for teaching [DeJong 91]. Among the possibilities of using the computer as a learning tool, it is an essential pedagogical tool allowing to teach topics that are difficult to illustrate with a traditional static method on a blackboard [Paget94], [Chataignier95]. As Dowling [Dowling96] writes : "Even of traditional types, they constitute 'Worlds' within which knowledge can be represented, explored and constructed". They must serve as a support for an intellectual work calling upon the student's imagination and allowing him to reach a better understanding of the model.

The understanding of the basic principles underlying the execution of the processor instructions is the first step in computer architecture. Then it becomes possible to explain the more complex techniques (pipeline, memory hierarchy, et cetera) which are used to improve the processor performance.

The goal of this simulator is to facilitate the learning of the basic principles, by allowing a realistic experimentation.

The student input is a source - assembly level - program, that can be interactively edited in the simulation environment. The simulator contains a two pass assembler in order to generate binary code. It is possible to perform step by step execution in the source program or to use break points.

During the first step, only the instructions set of the processor and the registers externally visible are taken into account. The simulator performs an atomic execution of each instruction and visualize the state of registers after each instruction. The student may at any time modify the content of registers. This allows the student to understand the instructions representation in the memory, and to assimilate the mechanism of program execution and the concept of "program counter".

The next step is a fine grain simulation based on a step by step execution at cycle level. Each instruction is decomposed in micro-instructions and all internal registers supporting the microprogrammed execution are now visible.

The simulator allows the student to visualize the internal state and to make the processor run cycle by cycle. Moreover it is possible to modify the microprogram for instance add a new instruction in the instructions set, and verify its execution.

2. Choice of the simulated processor

Our simulator targets the MIPS R3000 processor which constitutes the reference of the RISC architectures in the 1980's. It is simple enough and therefore well adapted to present the basic principles. This kind of processor is often considered as a case study [Tanenbaum91].

As this tool is intended for initiation to computer architecture, the internal architecture does not correspond to the pipelined version manufactured and used in the 1980's. For this approach we have preferred to modelize and simulate a microprogrammed version, which is easier to understand than a pipelined one. This microprogrammed version was developed by the Architecture team of the LIP6 Laboratory (UPMC). It is used as a VLSI project for postgraduate students. The processor model keeps the two original working modes (kernel and user) as well as the mechanism of exception and interruption.

The memory is divided into two segments, kernel and user, each segment including three zones one for the code, one for the data and one for the stack. But hardware support for virtual memory (TLB) has been eliminated.

As shown in figure 1, the internal architecture consists of a data path with registers and operators, and a controller including a micro-instruction memory and a sequencer. The micro-instruction is made of eigth fields, six of which are controling the Operation Part and two the sequencer. A mnemonic is associated to each value of a field. We have then a micro-instruction symbolic representation which simplifies both understanding and editing.

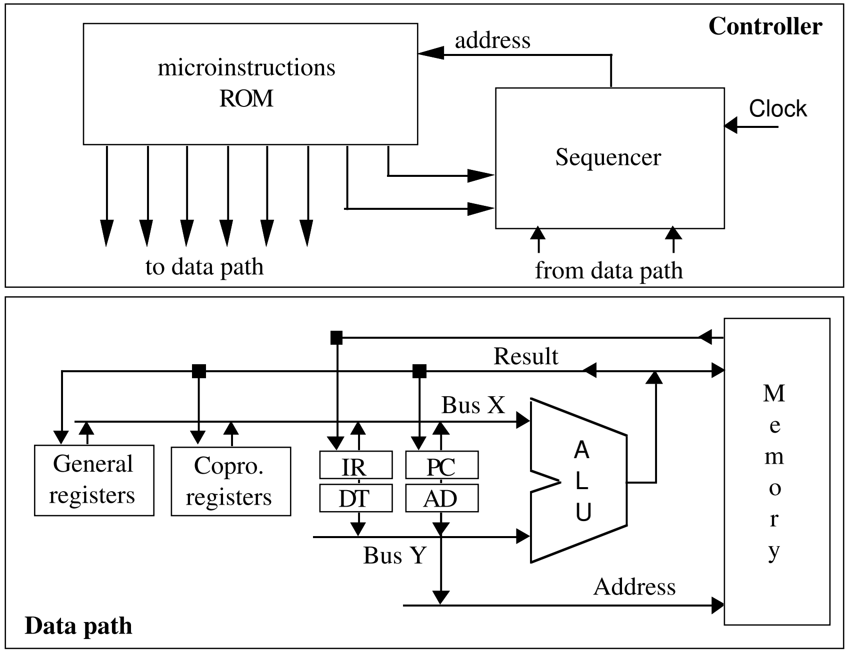

Figure 1 : MIPS R3000 Architecture

The choice of the MIPS R3000 allows the student to understand the mechanism of program execution such as the notions of exception and interruption.

A previous MIPS R3000 simulator, named SPIM, has been developped by James R. Larus [Larus] from university of Wisconsin and is presented in the book of Patterson and Hennesy [Patterson93]. SPIM is an instruction level simulator. Its main goal is to debug or analyse assembly level software for the MIPS R3000. The focus is on the speed and SPIM considers the processor as a black box.
SIMIPS is a step by step cycle-level simulator that focuses on the internal hardware architecture. Those two simulators are complementary and can then be used in a cooperative way.

3. Internal Architecture Model

The simulator is running on PCs under Windows NT. It has been written in Pascal using the programming environment Borland Delphi.
To develop the simulator, the object oriented approach conception method has been used. Basic objects such as registers, bus, ALU, memories,... have been defined. The data path is build from basic objets taking into account the hardware structure. The simulation of a processor clock cycle, which corresponds to the execution of one micro-instruction comes down to seven methods performing the following actions :

- reading of the micro-instruction in the ROM
- sending the six fields of the micro-instruction, to the data path, which transmits them to the basic objects
- sending of the sequence fields to the controller who transmits them to the sequencer
- propagation of the signals in the data path : the source operands are placed on the buses used by the ALU, the result is placed on the ALU output

- propagation of the signals in the controller : evaluation of the address for the next micro-instruction from the sequence fields received and if necessary from states coming from the data path
- writing in the data path : the result from the ALU is stored into the destination (register or memory cell) specified by the micro-instruction
- writing in the controller : the next micro-instruction address is written into the micro-instruction address register.

Such an organization is suited to the step by step mode micro-instruction by micro-instruction. The step by step mode instruction by instruction consists of executing a sequence of micro-instructions corresponding to the incrementation of the Program Counter, to the execution of the current instruction, the reading of the next one. In this mode the control is given back to the user when the micro-instruction that reads an instruction in the memory has been executed. The switching from a mode to another is possible at any time. While the user is in micro-instruction mode and if he wants in the middle of the instruction execution to switch for the instruction mode, the activation of a simulation step leads exactly after the reading of the next instruction. During the experimention, this flexibility of switching modes allows us to execute quickly instruction by instruction, then we can switch into the micro-instruction mode at the desired point of execution, for instance to test an additional instruction, the branching to an exception.

4. Graphic User Interface

The design has been done taking care about providing an environment with the maximum flexibility of usage.
The interface offers several tools : text editor, assembler, micro-instructions editor, simulator. Each tool is accessible through a separate window on the screen.
The figure 2 shows the global organization. The student can enter a program using an external text editor or the on-line text editor. This program is then translated by the assembler. The code generated by the assembler is loaded in the simulator, some registers are initialized, the simulation can begin.

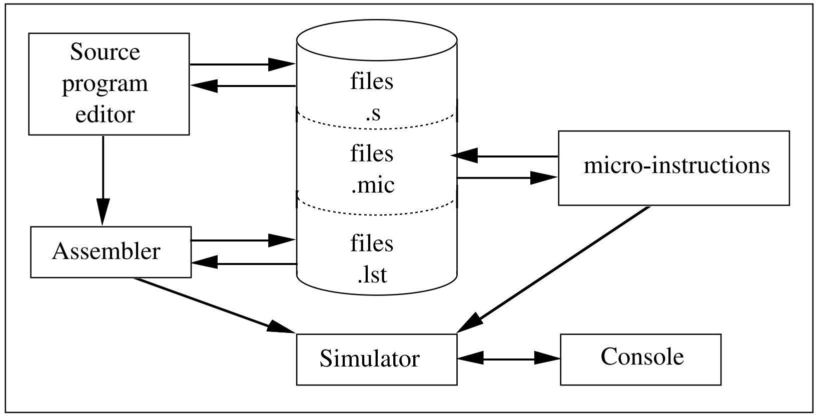

Figure 2 : Global view

4.1 The Text Editor / Assembler Window

As shown on figure 3, the role of this window is to allow the capture of the source program in assembly language. The window provides a command to assemble the source files into object files. Moreover it is possible to have an on-line help, to consult the specifications of the microprocessor assembly language. This on-line help gives the list of the directives, macros, mnemonics available and the syntax of the system calls provided by the simulation environment (read/write an integer in hexadecimal, write a string and quit the program).

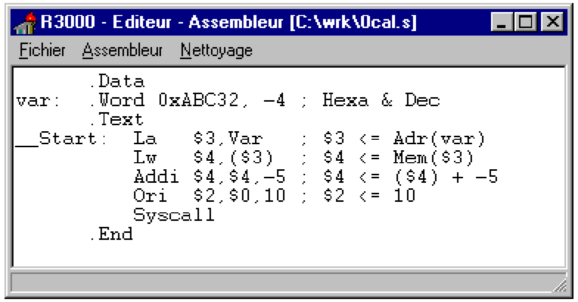

Figure 3 : Assembler window

4.2 The generated code window

In this dedicated window appears the generated code. The figure 4 shows a part of the one corresponding to the source program presented on figure 3. One can find the memory addresses with the coding of the instructions.

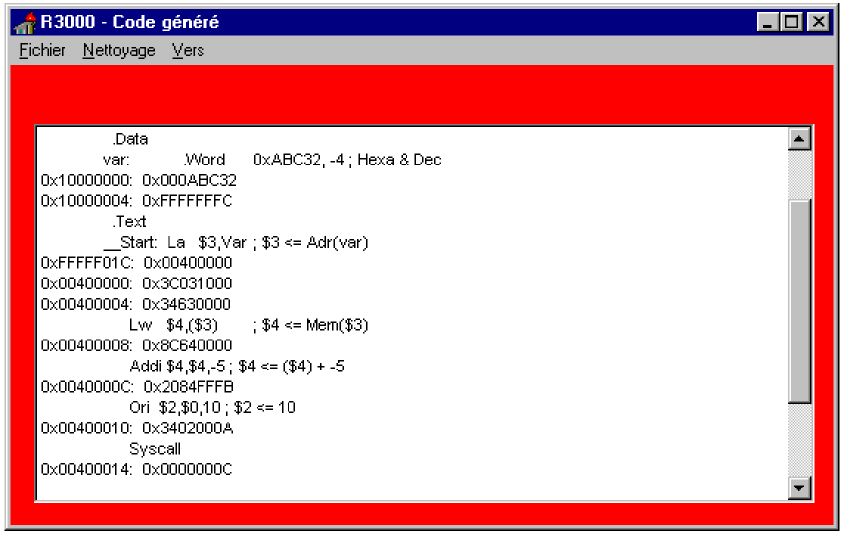

Figure 4 : Binary code window

4.3 The Simulation window

It is the main window, which is presented differently according to whether one is in micro-instruction or instruction mode (figures 5 and 6).

Whatever is the operating mode, at the center of the simulation window are displayed (figure 5) :

- the 32 general registers of the processor, as well as the dedicated registers, the Program Counter PC, the Bad Address Register BAR, the HI and LO registers, the EPC register used in case of exception, the status SR and the cause register CR. The content of each one can be modified at any time by the user
- the last executed instruction and the next to be executed
- the external events : the reset and the six hardware interrupts.

At the bottom is a display of the memory content : code, data and stack for the user, code, data and stack for the system.

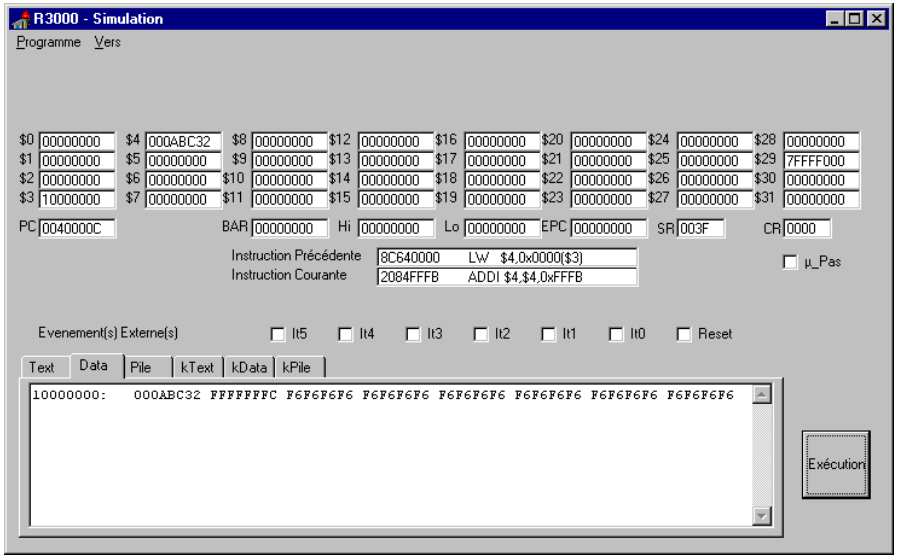

Figure 5 : Simulation window (instruction level)

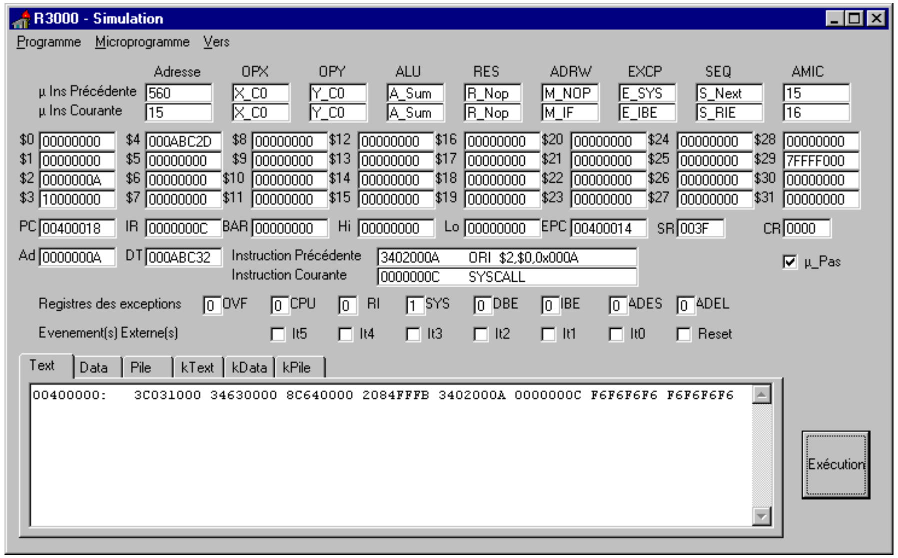

Figure 6 : Simulation window (micro-instruction level)

In micro-instruction mode a menu "Microprogram" appears, allowing to select a specific microprogram ROM. By default a standard microprogram ROM is used. In the window at the top of the registers (figure 6), are visualized : the last executed micro-instruction and the next micro-instruction to be executed.

In this cycle precise mode, all internal registers are displayed : AD, IR, DT such as the eigth exception flip-flop : OVR for arithmetic overflow, CPU for inaccessible coprocessor, RI for illegal opcode, SYS for syscall, DBE Data Bus Error, IBE Instruction Bus Error, ADES illegal address write, ADEL illegal address read.

4.4 The Journal and Console windows

These windows are present to facilitate the work of the student.
The Journal window displays an history of the instructions or micro-instructions executed according to the simulation mode.
The Console window visualizes only the input/output corresponding to the system calls during the program execution until its end.

4.5 The Micro-instruction's Editor window

This window allows changes in a microprogram ROM. In the upper part, is a display of all the micro-instruction fields, each of them with a pop-up menu to select a mnemonic. The address defines which micro-instruction will be changed.

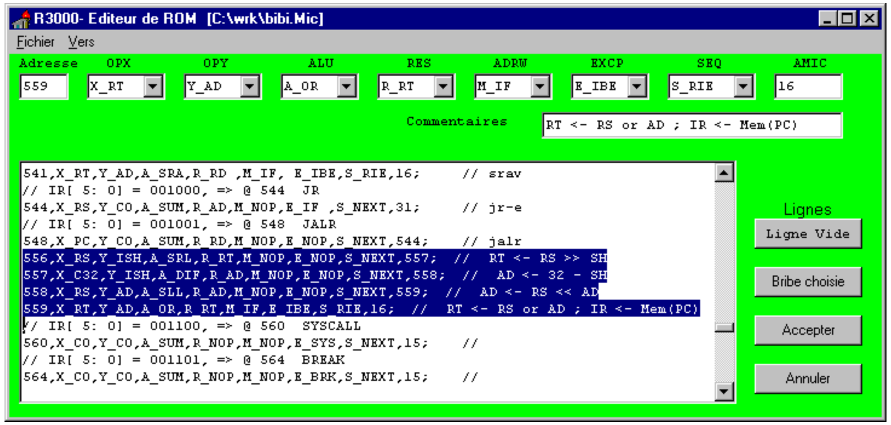

Figure 7 : Micro-instruction's Editor window

5. Experimentation

SIMIPS has been used since October 1997 by 150 under graduate students in computer science at the Pierre et Marie Curie University in Paris. The teaching begins with the R3000 processor's external architecture and assembly language. When the bases are introduced, the study goes on in the microprogrammed mode. After a quantitative evaluation of the students results, it appears that our goal, that was to induce an intellectual work leading the students to gain more understanding of the internal execution of the processor has been reached.

6. Conclusion

This MIPS R3000 processor's simulator SIMIPS is easy to use because of its graphical interface, ergonomic and pedagogic. It represents an interesting conceptual help for the students to assimilate the processor basic internal organization as well as the general principles of microprogramming. The interest of the student is focused on what happens cycle by cycle. Moreover the possibility to add and to modify micro-instructions, permits him by experimentation, to verify himself his good understanding of the processor micro-architecture.
This tool is not theory-based but represents a learning by doing approach. Its goal is to facilitate and complete a classical teaching by increasing its pedagogical efficiency.

References

[Tanenbaum91] Tanenbaum A. : *Architecture de l'ordinateur,* Troisième Edition, InterEditions, pp. 601-618

[De Jong91] De Jong T. : *Learning and Instruction with Computer Simulations,* Educational Computing, N°6, 1991

[Larus] Larus J.R. : SPIM S20 : A MIPS R2000 Simulator, Report 25p, ftp.cs.wisc.edu/pub/spim.

[Patterson93] Patterson D., Hennessy J. : *Computer Organization and Design. The Hardware/Software Interface,* Morgan Kaufman Publisher.

[Paget94] Paget M-M., J-F. Couturas, D. Robic, A. Tarrago : *Graphical animations for the teaching of recursivity,* CALISCE94, International Conference on CAL and Instruction in Science and Engineering, Paris (France), pp. 285-292

[Chataignier95] Chataignier P., F. Madaule : *Multiproc : une simulation pédagogique d'un multiprocesseur,* Environnements Interactifs d'Apprentissage avec Ordinateur, Tome 2, Eyrolles, Paris

[Dowling96] Dowling C. : *Simulations- New 'Worlds' for Learning?* Educational Multimedia and Hypermedia, AACE edited by P. Carlson and F. Makedon, pp. 196-201