Simulation Based Learning of Microprogrammed Processor

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

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

Francois.Dromard@lip6.fr
Marie-Martine.Paget@lip6.fr

http://www-asim.lip6.fr

Abstract

We use the non-pipelined version of MIPS R3000 microprocessor to display the execution mechanism of instructions. The SIMIPS simulator has been developed to improve the understanding of this mechanism. SIMIPS shows the internal behaviour of the processor, instruction by instruction as well as micro-instruction by micro-instruction. It allows to modify the microprogram in order to introduce new instructions.

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

1. Introduction

An effective teaching of computer architecture for beginners is a tough job and traditional methods are not well suited to understand how effectively computer works during the execution of a program.
In computer architecture we must present how the processor works at various levels, from instruction level to micro-instruction level. Simulations at register transfer level are useful to show up the instruction processing as well as the detailed mechanism of exceptions and interrupts handling.
Numerous researches have been carried out in this field. Some authors better present architecture and assembly language for a real processor [Devore87], while others rely rather on a hypothetical processor [Cutler90] [Koneva90]. For Waldron [Waldron98] MIPS is "the preferred choice for teaching computer architecture in the 2000s, just as the Motorola 68000 was during the 1980s". So we chose the MIPS R3000, a real 32 bits processor, supporting both user and kernel mode for modern operating systems, easy to learn and to understand. Besides, at Pierre et Marie Curie University, this processor is a basic tool used in several "modules" of a Master Degree (computer architecture, compiling techniques and VLSI design).

2. Targets

The SPIM simulator [Laurus] allows to assemble and debug assembly code written for the MIPS instructions set. SPIM is very helpful to allow students to be familiar with the computer architecture and capable of writing numerous small programs in asssembly language.
With the SIMIPS simulator, we realized a versatile tool showing different levels of instruction execution. In the first step we show the instruction execution mechanism at the level of transfer between programmer's registers. In the second step, we split up each instruction into micro-instructions and we make visible the internal registers used at the micro-instruction level execution. These two levels are described in [Body98]. The input file is one assembly level source program. SIMIPS contains a built-in assembler. As it contains also a macro-assembler, the source program can also use macros defined by the programmer. Finally we developed a graphical interface, as close to the internal architecture as possible, since it allows to see the data transfers from one place to another (from a register to another register or to a memory cell).
For Shankararaman [Shankararaman98], three strategies (tutoring by example, cognitive apprenticeship et successive refinement) must complete visual animations. We had not the same approach because we think that a student must have an opportunity to interact freely with the processor model. The cognitive apprenticeship strategy allows him to play an active role in the learning process.

3. Recall about the internal architecture

The chosen version of the MIPS R3000 is not pipelined, for the sake of simplicity and because most dedicated processors for industrial applications are still microprogrammed. The classical CISC architecture (data path and microprogrammed controller) is much more general than the sophisticated multipipelined architecture existing in today's high-end superscalar microprocessors. The internal architecture consists of a data path and a control part which includes a sequencer and a micro-instructions ROM. The control part determines the transfers to be done on the data path. As shown on Figure 1, the data path is composed of an ALU, a register bank and buses X and Y. Within a clock cycle, it achieves the elementary transfers of data between registers or memory cells. Note that the memory is not considered as a part of the processor.

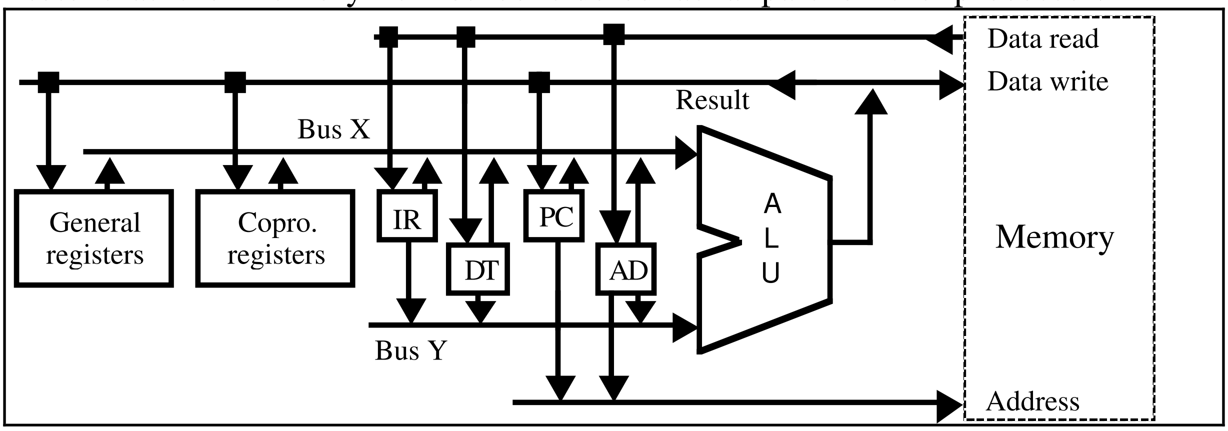

Figure 1 Data Path

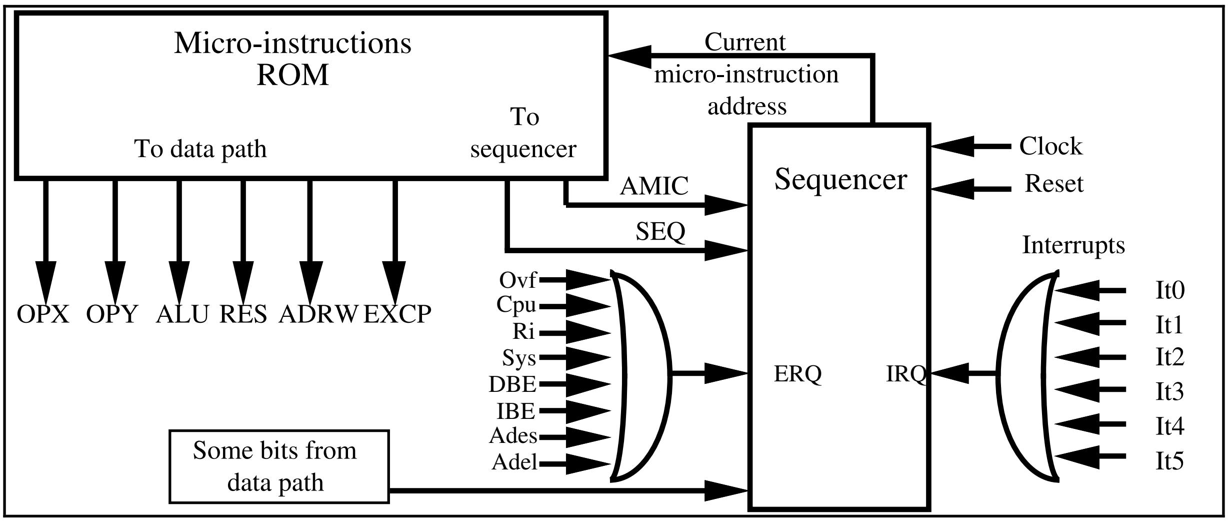

Figure 2 Control Part

Figure 2 shows the control part. The ROM generates the control signals to the data path and the control signals to the sequencer. The first ones are organized in six fields OPX, OPY, ALU, RES, ADRW, EXCP, the second ones in two fields SEQ, AMIC. They compose the eight fields of a ROM word :

Bus X	Bus Y	Operation	Writing	Memory	Exception	Sequence	Adress
OPX	OPY	ALU	RES	ADRW	EXCP	SEQ	AMIC

The sequencer generates the next micro-instruction address, according to the fields SEQ and AMIC, the data path elements, the exceptions and external events (interruption, reset).
With such a representation, the student can see that the control part is a finite state machine (FSM), where the current micro-instruction address represents the FSM state, the next micro-instruction address being the next FSM state and the micro-instruction itself being the FSM outputs.

4. Simulation at instruction level

The simulation displays the internal behaviour of the processor, instruction by instruction. To execute a program, one must edit the source program in assembly language (file.s) by using the editor window and assemble it (file.lst) in order to get an executable code. Then the user can load the code into the simulator and execute it. In the simulation window, one can see the current and previous instructions, the 32 general register contents (R0 to R31), the dedicated register contents, the external events, reset and interruptions, as well as the memory content. During the program execution, the student can modify a register content (he cannot modify the memory content in the current version). A journal window displays the program trace.

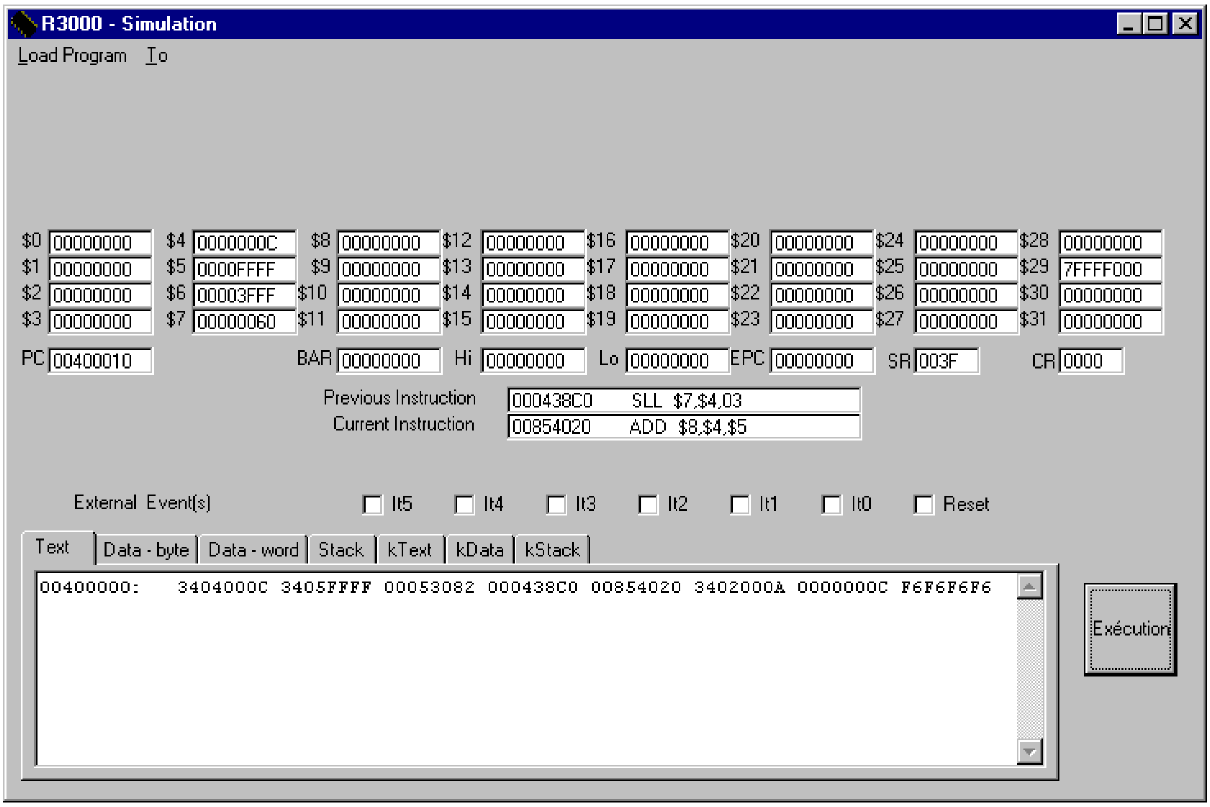

Figure 3 Simulation instruction by instruction

Figure 3 shows the instruction to be executed :(`add $8,$4,$5`), in the following source program :

```
.text
ori    $4,$0,12     ; R4 <- 12
ori    $5,$0,-1     ; R5 <- -1
srl    $6,$5,2      ; R6 <- (R5) >> 2
```

```
sll    $7,$4,3        ; R7 <- (R4) << 3
add    $8,$4,$5       ; R8 <- (R4)+(R5)
ori    $2,$0,10       ; parametre de fin
syscall               ; Termine le programme
.end                  ; fin du fichier
```

5. Simulation at micro-instruction level

This simulation shows the detailed processor behaviour cycle by cycle, so that the student could check by himself how the processor works.
In the micro-instruction mode, the 3 internal registers (IR, AD et DT), and the 8 exception flip flops can now be seen in the simulation window. The control part is not shown entirely. The current micro-instruction, the previous micro-instruction and the next micro-instruction address computation only are visible. We use mnemonics for each micro-instruction field to represent the values in a symbolic way.

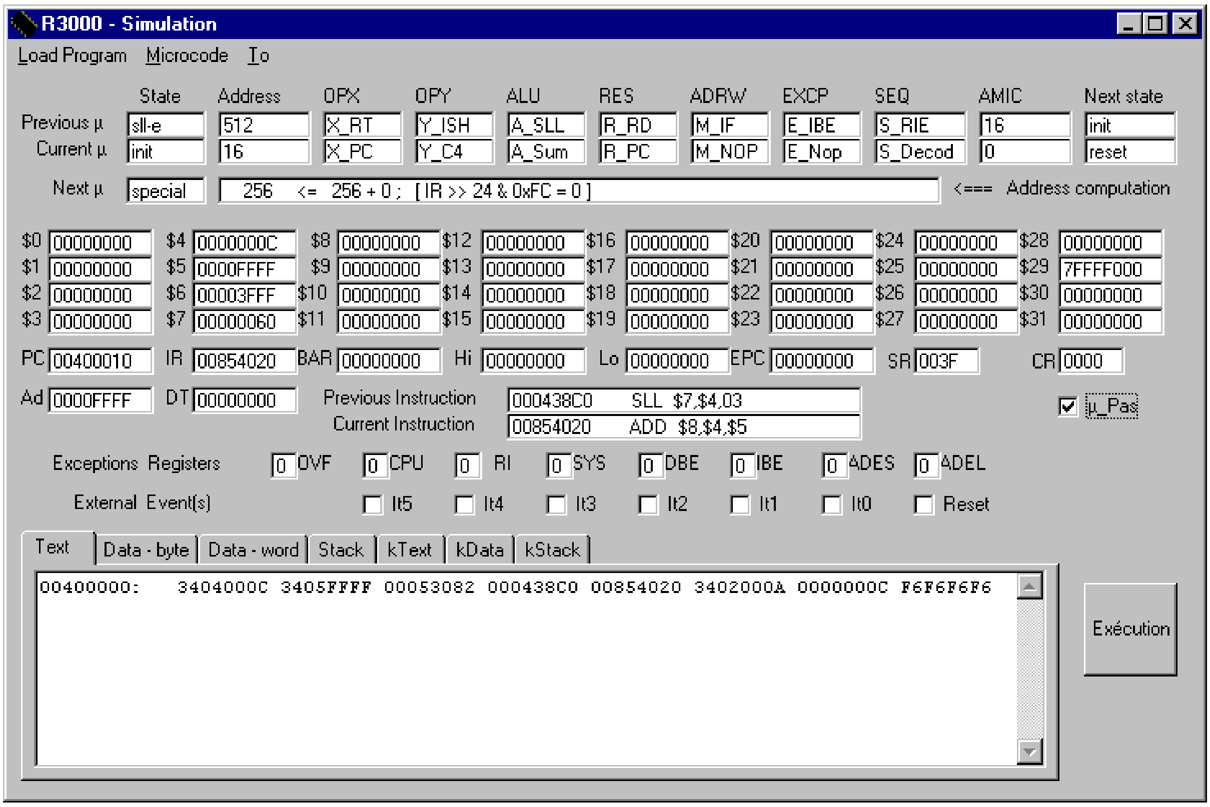

Figure 4 Simulation micro-instruction by micro-instruction

Figure 4 shows the first micro-instruction performing the execution of the following instruction :

```
add $8,$4,$5
```

This mode gives one the opportunity to see what happens when an exception occurs. A micro-instruction editor window is available and allows the student to change the microprogram. Then it is possible to microprogram new instructions.

6. Macro-instructions

A macro-instruction (or macro) is written when a given sequence of instructions in a program is repeated. To define a macro, we create a header containing its name, a body and the end delimiter. Whenever the macro is used, the assembler replaces the call by its body. In the first step, the assembler replaces the macros included in the source text by MIPS instructions. The second step consists of generating an executable code. Our simulator accepts macros with formal parameters, so that students studying compiling techniques may generate macrocode that can be executed by the simulator. So they can validate the macrocode.

7. Global microprogrammed animation

As we said before, our simulator allows a student having a certain degree of experience to watch the MIPS processor behaviour at a very low level of architecture. Moreover the student can see how the processor performs a single micro-instruction in a single cycle and how data flows throughout the data path.

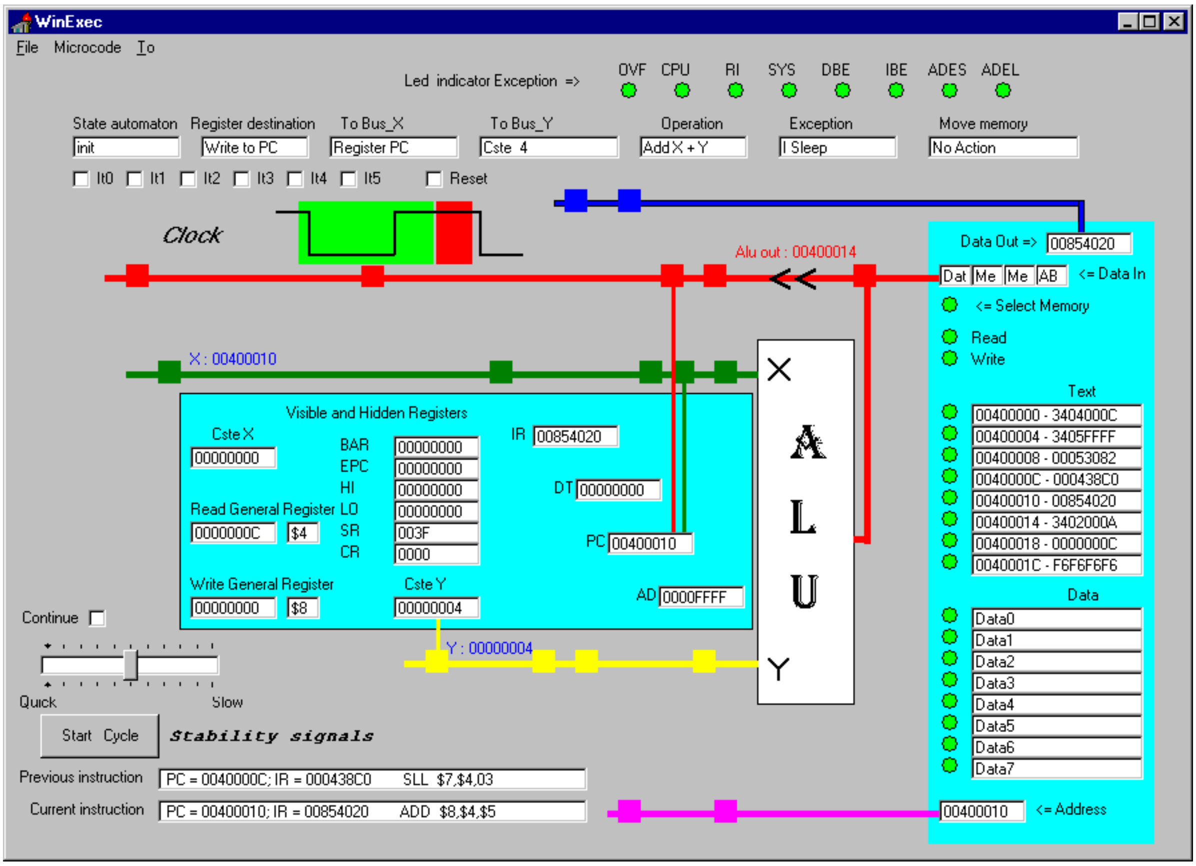

Figure 5 Global microprogrammed view

On figure 5, we can see how the micro-instruction execution is represented. For each micro-instruction, we display the components involved in the execution (registers, buses,...) and, on the right of the window, appears the memory. Above the data path, is displayed the current micro-instruction and the meaning of the fields. The user can follow step by step the information transfers during the micro-instruction execution. This demonstrative view can improve the understanding of the execution mechanism, by focusing on what is really used at any time, but it does not replace in any case the full exhibition described in section 5 on Figure 4.

8. Experimentation

The MIPS R3000 processor is widely used at the Pierre et Marie Curie university for teaching purposes, all along the studies in Computer Science. Since October 1997, more than 150 undergraduate students use the simulator each year. They appreciate this tool, which leads them to a better understanding of the internal mechanisms. This course is delivered to (BAC + 3) students. It is completed by a more advanced course (BAC + 4) where the RISC pipelined architecture is presented as well as features such as caches and virtual memory management. Then the VLSI implementation of the MIPS R3000 processor is done in the framework of a postgraduate course (BAC + 5) from the VHDL specification to the drawing of the masks.

9. Conclusion

In SIMIPS, each view (instruction level, micro-instruction level, data path level) corresponds to a different abstraction level. Each window provides interactivity, with a graphical interface making visible the actions executed within the machine. These features can be observed in three different modes : instruction by instruction, micro-instruction by micro-instruction or the animated data path. The originality of this work relies in the variety of the approaches that have been chosen to present the concepts. The SIMIPS simulator is built to execute MIPS R3000 processor code. To facilitate its use, the simulator contains a source text editor (mnemonics and macros), a macro-assembler, an assembler, a journal to keep the execution history and a console to present the program input/output.

References

[Devore87] Devore J. J. and Hardin D. S. : *A computer Design for introducing hardware and software concepts,* IEEE Trans. Educ., Vol 30, pp. 219-226

[Koneva90] Koneva L. and Denev J. : *EASY/VI A new instructionnal computer,* ACM SIGCSE Bull., Vol 22, N°2, pp. 55-58

[Cutler90] Cutler M. and Eckert R. R. : *Microprogrammed computer simulator tools,* IEEE Trans. Educ., Vol 33, pp. 212-220

[Body98] Body Y., Dromard F., Greiner A., Paget M-M., Pétrot F. : *SIMIPS : A cycle precise interactive simulator for teaching microprocessor architecture*, CALISCE98, 4th International Conference on Computer Aided Learning and Instruction in Science and Engineering, Göteborg (Sweden), pp. 146-154, ISBN 91-7197-683-3

[Shankararaman98] Shankararaman V., Munoz J.M., Banks E. : *Simulation Based Learning of computer architecture,* CALISCE98, Göteborg (Sweden), pp. 424-432

[Waldron98] Waldron J. : *Introduction to RISC Assembly Language Programming,* Addison-Wesley, pp. 1-180, ISBN 0-201-39828-1

[Laurus98] Laurus J. R. : laurus@cs.wisc.edu, ftp.cs.wisc.edu/pub/spim