UCOMIPSIM 2.0: Pipelined MIPS Architecture Simulator

Andrés Gersnoviez, María Brox, Miguel A. Montijano, Juan A. Sújar and Carlos D. Moreno
Dept. Electronic and Computer Engineering
Escuela Politécnica Superior, Universidad de Córdoba
Córdoba, Spain
[andresgm, mbrox, el1movim, i22suroj, el1momoc]@uco.es

***Abstract*—The objective of this work is the realization of an application in JAVA whose function will be to simulate the MIPS 32-bit architecture in pipelined computer with hardwired control unit, in order to help in the learning and understanding of this architecture by the students in Computer Architecture subjects. With it, students will be able to graphically visualize the datapath of multiple MIPS instructions passing through the different stages of the pipelined computer.**

Keywords—MIPS, Pipelined Computer, Teaching Simulator, Computer Architecture.

I. INTRODUCTION

The use of simulators has demonstrated that help the students to reach higher levels of comprehension on the subjects [1]-[6]. Specifically, the study of pipelined computers is difficult for the students of Computer Engineering; for this reason, the development of teaching tools that help to achieve a good understanding of pipelining dynamics, is very important. Nowadays there are applications that help with the learning of this architecture [7]-[13]. Among these applications one of the most relevant tools is MARS simulator [7]-[8]; MARS is a very good MIPS code execution simulator, but it lacks graphical visualization of the datapath. Other remarkable simulator of pipelined computers is MIPSim [9]; this simulator has graphical visualization of the datapath, but it lacks the *forwarding* and *hazard detection* units [14].

To solve the lack of these units, the teaching area of Computer Architecture at the University of Cordoba developed an earlier version of this simulator presented in this paper [10]. This first version has *forwarding* and *hazard detection* units, but it uses obsolete software and has an incomplete datapath that led to execution errors in some instructions.

Therefore, a new simulator based on the Java architecture of Sun Microsystems has been developed. This new teaching tool corrects the errors of the earlier version, as well as including a much wider instruction set.

The structure of the article is as follows. A brief summary of the MIPS pipeline, data and control hazards is included in Section II. The graphical interface of the UCOMIPSIM simulator, as well as its instruction set and some execution examples are presented in Section III. The reception of the simulator by students and their feedback is summarized in Section IV. The improvements that are expected to be included in future versions of the application are shown in Section V. Finally, the article concludes with Section V for conclusions.

II. MIPS PIPELINE

The pipeline strategy is based on separating the instruction datapath into a series of consecutive stages. In this way, different instructions can be executed at the same time. Each instruction passes sequentially through all the stages, so that each stage is executing a different instruction. Specifically, MIPS pipeline is composed of five stages: IF (instruction fetch from memory); ID (instruction decode), phase where the registers are read and the control signals of the instruction are generated; EX (instruction execution), where the result of a R-type operation or the address used for a branch or data memory access are calculated; MEM (memory access), where the data memory is accessed to read or write; and WB (write back), where the result is stored in the destination register [14].

The different stages are separated by registers where the data and control signals that will be necessary in later stages are stored. The name given to these registers is the composition of the stages that separate. Thus, IF/ID register is the one that separates IF and ID stages.

Pipeline does not improve the latency of instructions (number of cycles that pass from the time an instruction is searched until it is executed). In fact, it aggravates it, since most instructions do not need to go through the five stages of pipeline and it is imperative that these instructions go through all of them. However, the throughput is greatly improved, achieving, under ideal conditions, the performance of one instruction per clock cycle.

Emphasizing a commentary from the previous paragraph, conditions will not always be ideal, because conflicts may appear between instructions that are going through the pipeline. It may be that an instruction needs as source register a register that is going to be modified by an instruction that is still in the pipeline; or it may be that an instruction is going to change the program execution course and some instructions have already advanced in the pipeline that are not going to be executed in the new course. This is what is known as

978-1-5386-0928-6/18/$31.00 ©2018 IEEE

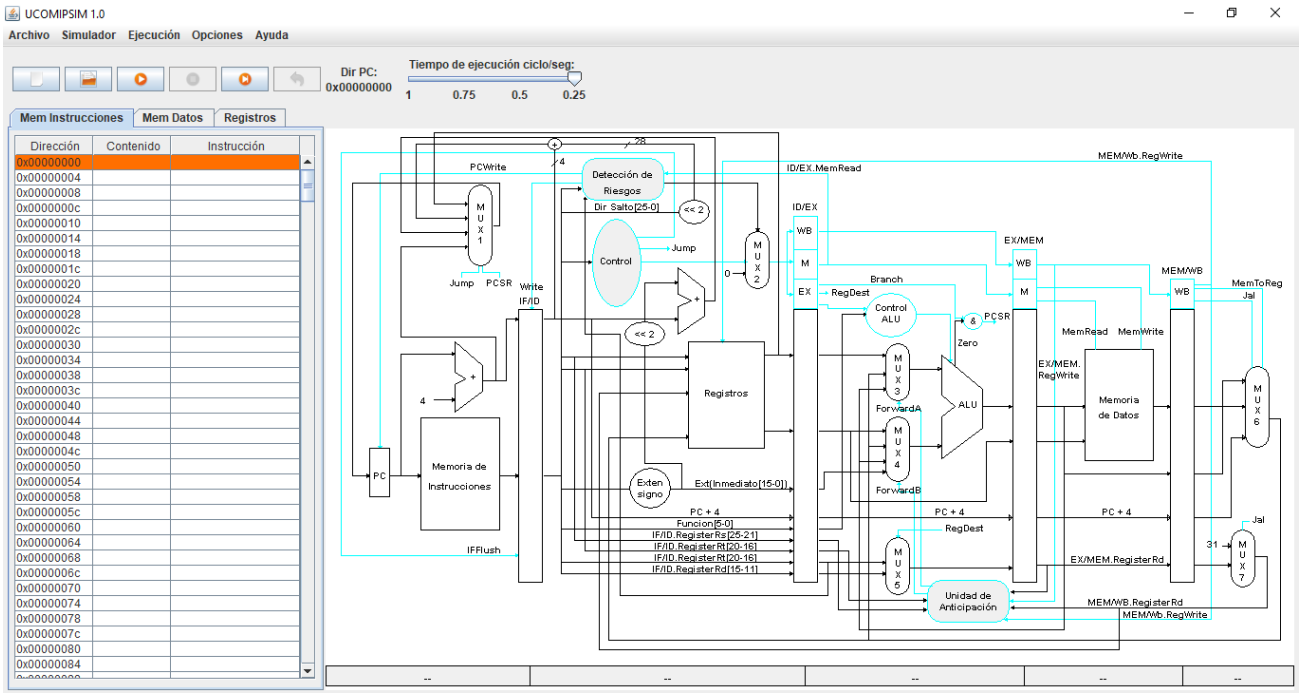


Fig. 1. UCOMIPSIM main window

"hazards" and there is a distinction between data and control hazards.

A. Data hazards

This is the case of instructions that use as source register a register that will be updated by an instruction that has not yet been executed. An example of this can be seen below:

```
add $1, $2, $3
sub $4, $5, $1
sw $6, 8($1)
```

As it can be appreciated, sub and sw instructions need $1 register for their calculations, but it will not be updated until add instruction is finished, having to stop, in principle, these two instructions until add instruction is finished.

The reality is that these instructions do not need to be stopped, because the new value of $1 can be known before add is finished. Specifically, the new value of $1 is calculated in the EX stage of add instruction and it is transferred to EX/MEN and MEM/WB registers when it passes through the pipeline. On the other hand, sub and sw really need the new value of $1 is in their respective EX stages. Therefore, $1 source register used by the ALU in sub can be replaced by the destination register in EX/MEM and, in the case of lw, replace it by the destination register in MEM/WB.

For this reason, the *forwarding* unit is introduced and is used for detecting the dependencies between the registers in the pipeline, as well as replacing a source register of an instruction with the destination register stored in EX/MEM or MEM/WB when it is applicable.

However, the *forwarding* unit does not solve all the cases. If instead of being add the first instruction, it would have been lw, also using $1 as destination register, the *forwarding* unit solves the problem for sw, but not for sub, since the updated value of $1 will not be available until MEM stage of lw is finished. In other words, sub needs to wait for a cycle in order to access the correct data.

For these cases, the *hazard detection* unit is introduced. Specifically, when this happens, the instructions that are in stages IF and ID (sw and sub, respectively, in our example) are blocked for one cycle. In the EX stage a "bubble" is introduced, in which nothing is done, assigning all its control signals associated to zero, and it will be transferred along the pipeline until disappearing.

B. Control hazards

The other type of hazard happens when a “branch” instruction is presented. In this case, depending on a comparison of registers developed in EX stage, it is determined whether the program execution is still sequential or a branch happens. The hazard is created because the instructions that, at the time, are in IF and ID stages may not be correct.

In theory, MIPS assumes by default that the branch will not happen, leaving in the previous stages the instructions that follow in sequential order. In this way, if the branch does not happen, the throughput is not affected. However, when the branch happens, then those following instructions found in the pipeline must be discarded.

For this, the *hazard detection* unit has to be updated. Specifically, when the branch condition happens, a bubble must be introduced in the ID stage, while IF instruction is replaced.

III. UCOMIPSIM Simulator

UCOMIPSIM is an application developed in Java, which simulates the datapath explained in the previous section, including *forwarding* and *hazard detection* units. The image of the simulator graphical interface can be seen in Fig. 1. The

TABLE I. UCOMIPSIM INSTRUCTION SET

Instruction Type	Instructions
R	add, addu, sub, sll, srl, and, or, nor, xor, not, jr, slt.
I	addi, addiu, lw, sw, lb, sb, slti, sltiu, lui, beq, bne.
J	j, jal.
Pseudoinstruction	move, bgt, bge, blt, ble.

MIPS pipelined datapath is shown in it in a self-climbing image. The graphical interface includes the menu bar and toolbar at the top of the figure; and the instruction memory, data memory and registers can be accessed at the left of Fig.1

In the menu bar, the memories and registers can be initialized (by assigning them to zero), load a file with a program to simulate; to save the results of a program after it has been executed; to start a simulation, pause it, execute a simulation step by step or restart it; on the other hand, the starting values of the addresses of the instruction and data memories, as well as the initial value of the PC, can be configured; finally, an user manual of the application is available in the help.

The toolbar is composed of buttons that give direct access to the most frequently used menu actions. These are memory initialization, file loading, simulation startup, simulation pause, step-by-step simulation and simulation restart. There is also a bar to regulate the execution speed of the simulation.

A. Instruction set

UCOMIPSIM can operate with a short subset of instructions, but large enough for teaching purposes. The complete set is shown in Table I.

In order to be able to operate with this instruction set, the datapath explained in [14] has had to be expanded, especially for instructions such as j or even jal, as it will be seen in a later example. This decision was adopted to give a more detailed view of how MIPS computer works.

As it can be seen in the table, a number of widely used pseudoinstructions have also been included. When a file with pseudoinstructions is loaded into the simulator, the simulator translates these pseudoinstructions into computer instructions, just as the assembler would do it. In this way, students can better understand what pseudoinstructions actually become when these are executed by the computer.

B. Execution examples

When the execution of a program is simulated, instruction and data memories and the register bank can be manipulated directly from the simulator interface. However, a text file with the program data can also be created and loaded via the simulator. An example of this type of files is shown in Fig. 2. As it can be seen, the initializations of the instruction memory addresses (where the first program instruction begins), of data memory, PC and registers, can be specified in the file. Once this part is finished, the program is specified and comments can be included by using the "//" symbol.

Once the program is loaded, the instruction memory, placed on the left side of the interface, shows it. This memory illustrates the address of each memory word, its contents (both data in hexadecimal code) and the instruction corresponding to that code.

As it can be seen in the example of Fig. 2a, there are numerous data and control hazards.

One of the execution cycles of the program in Fig. 2a is shown in Fig. 3. First, in the instruction memory, it can be seen that bge pseudoinstruction has been converted into a slt instruction followed by a beq instruction, placing the registers to compare in the appropriate order.

In the figure it can be seen how each instruction has an associated color and the corresponding datapath for each one is highlighted in the same color.

In this case, sub is in WB stage, close to its end; sub is followed by slt, which is in MEM stage; beq is in EX stage; in ID stage there is a bubble; and, finally, srl is in IF stage.

Firstly, the data hazard between beq and slt must be highlighted. To make the branch or not, beq needs the result of slt. In this case, the *forwarding* unit provides the slt result and places it in the ALU (highlighted in green) to be processed by beq in its EX stage.

All the units and registers that appear in the simulator, just by clicking on them, will be open in a new window, showing a more detailed image of them. If in this cycle a click is done on the *forwarding* unit, a window will appear with the

```
// Initialization of memory addresses and PC
DIRECINSTRU[00000010]
DIRECDATOS[00000000]
PC[00000010]

// Initialization of registers:
$16 = 1
$17 = 4
$18 = 6
$19 = 20

// Data memory:
DATA[0000001C] = 14

// Program:
sll $8, $16, 1
add $9, $8, $17 // (Comments)
sub $10, $18, $9
bge $10, $0, 2
sll $9, $9, 3
add $12, $9, $8
srl $8, $8, 1
lw $10, 8($19)
```

(a)

```
// Initialization of memory addresses and PC
DIRECINSTRU[00000000]
DIRECDATOS[00000000]
PC[00000000]

// Initialization of registers:
$17 = 16

// Instrucciones:
move $16, $0 // i = 0
addi $18, $0, 5
sll $4, $16, 2 // a0=4i, label loop
move $5, $16 // a1=i
jal 13 // jal to rst2
sll $8, $16, 2
add $8, $17, $8 // t0=dir A[i]
sw $2, 0($8) // A[i]=v0
addi $16, $16, 1 // i=i+1
bne $16, $18, -8 // branch to loop
EMPTY
EMPTY
EMPTY
sub $2, $4, $5 // label rst2
sll $2, $2, 1
jr $31
```

(b)

Fig. 2. Examples of program files in UCOMIPSIM: (a) Example 1; (b) Example 2.

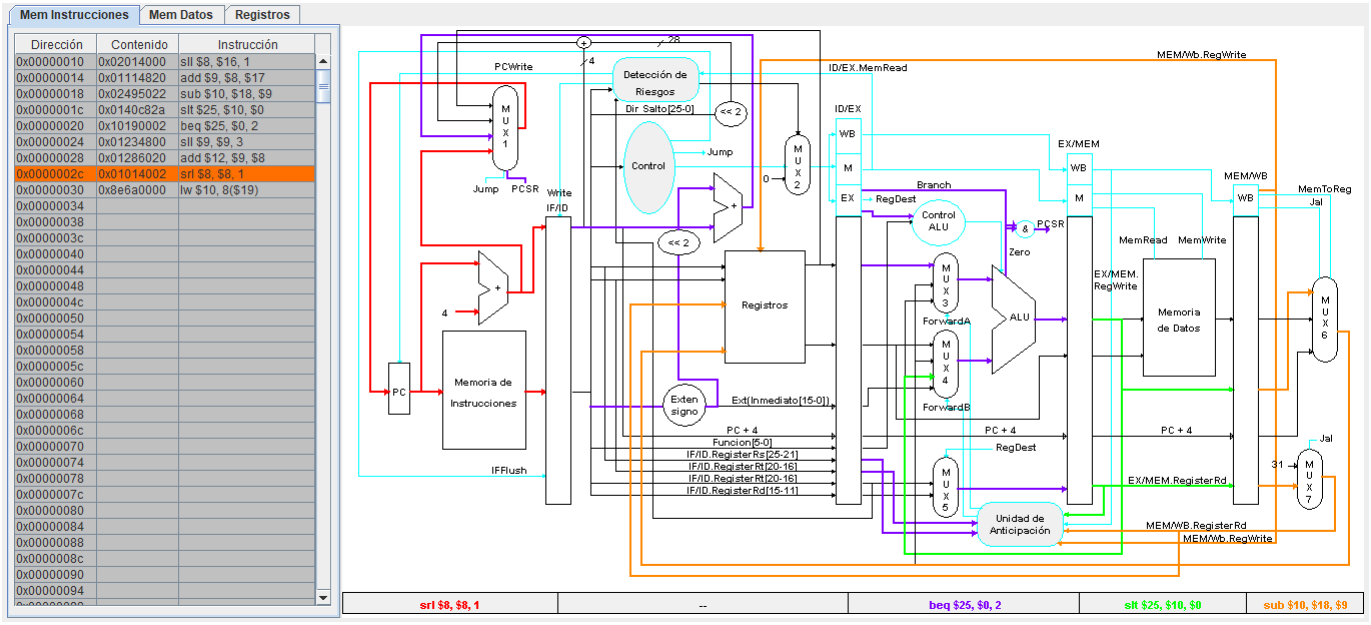


Fig. 3. Example of data and control hazards in a program execution in UCOMIPSIM

information of Fig. 4a. In this case it is possible to check the input and output values of this unit. In addition, the simulator allows to control the simulation course from this new window (start, stop, step-by-step or restart simulation). In the same way, a click on multiplexer 4 can be performed, checking its different input values and how, thanks to the signal provided by the *forwarding* unit (ForwardB), selects the value of rd register provided by EX/MEM register (Fig. 4b).

In second place, after beq is checked, there is a control hazard. Specifically, a branch must happen and the instruction that follows sequentially to beq (i. e. sll) is in ID stage. In this case, the hazard detection unit inserts a bubble into ID stage (as it is shown in Fig. 3), and replaces PC with the address calculated by beq, so the srl instruction appears in IF stage.

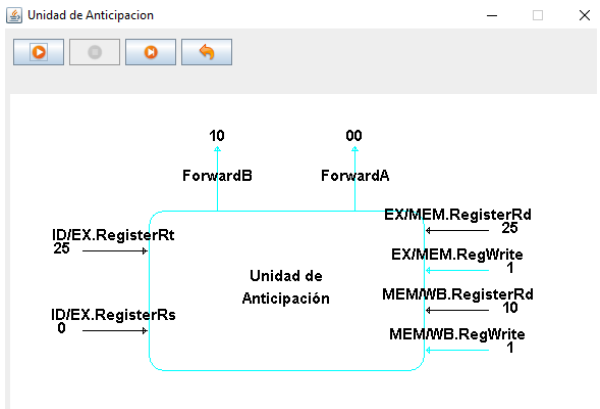
Finally, in addition to the instruction memory, the user can also analyze and modify the data memory and the register bank. This is shown in Fig. 3, next to the instruction memory, the tabs that give access to these sections. Once the execution of the program in Fig. 2 has been finished, the information of the data memory and the register bank is shown in Fig. 5a and 5b respectively.

In order to analyze the behavior of jal on this computer, the program in Fig. 2b is loaded into the application, as it can be seen Fig. 6. In this case, the datapath shown in [14] has been expanded. Specifically, PC+4 value has been propagated along the pipeline, at the same time that a control signal called "jal". When WB stage is reached, in addition to being able to choose between the result of the ALU and the data read from memory, the value of PC+4 is added. If jal control signal is active at this stage (as is the case in Fig. 6), multiplexer 7 will choose PC+4 as the data to save and the destination register will be $31, i. e., $ra.

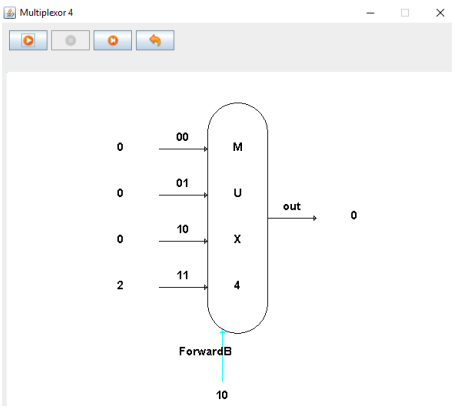
In the same figure, at the same time that jal is in stage WB, jr instruction is in ID stage. As it can be seen, the contents of $31 register are read and taken to PC, returning to the main program.

IV. UCOMIPSIM Reception and Feedback

Once a beta version of the application was finished, a version of it was provided to the students of the subject "Advanced Computer Architecture" of the degree in Computer Engineering at the University of Cordoba. The aim was to

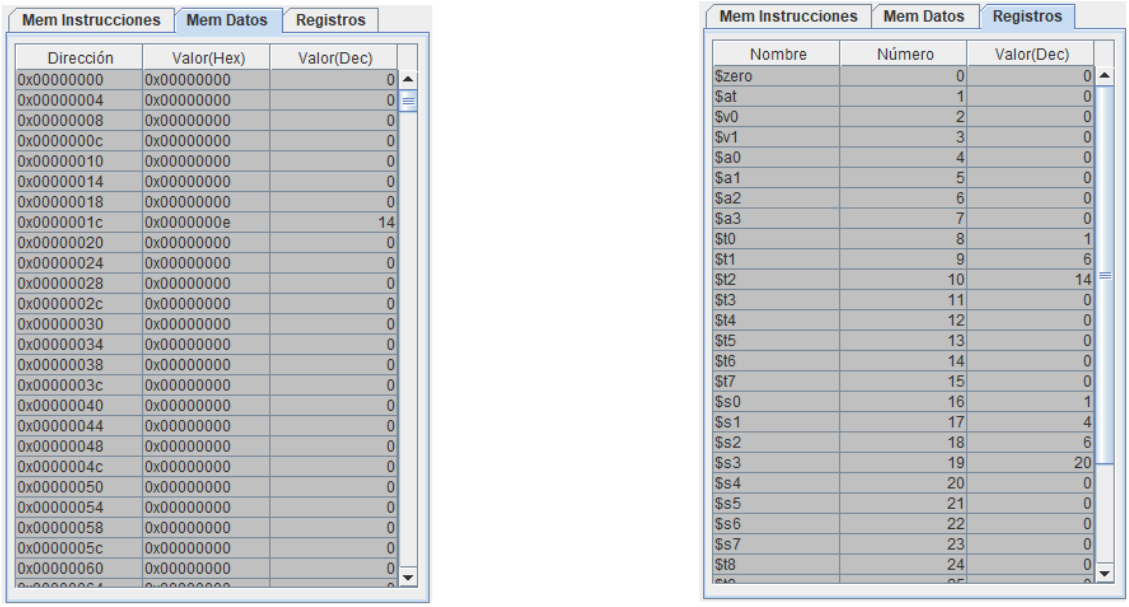


(a)



(b)

Fig. 4. Visualization of components in more detail (a) *Forwarding* unit; (b) Multiplexer 4



(a) (b)

Fig. 5. (a) Data memory; (b) Register bank

collect impressions and detect errors that had been ignored. In addition to the application, a template was provided where the students could describe their impressions, mistakes found and improvements on items that they would have liked to see in the application.

The impressions collected were very positive. Specifically, the students found it an user-friendly and intuitive application, greatly reinforcing the knowledge taught in the subject.

Among the proposed improvements, it was considered that the simulation could be executed at different speeds specified by the user; also that each time a component was clicked on, a new and independent window was opened with this component in greater detail, being able to have several windows of components open at the same time (initially it was a single window that was updated each time a new component was clicked); the students also proposed that it could be controlled from these same windows of components the simulation process; finally, they showed their interest to see the behavior of jal instruction (which was not included in the original instruction set of the application), which involved modifying the datapath presented in the application.

All these improvements were considered interesting and that could enhance the learning, so that these modifications

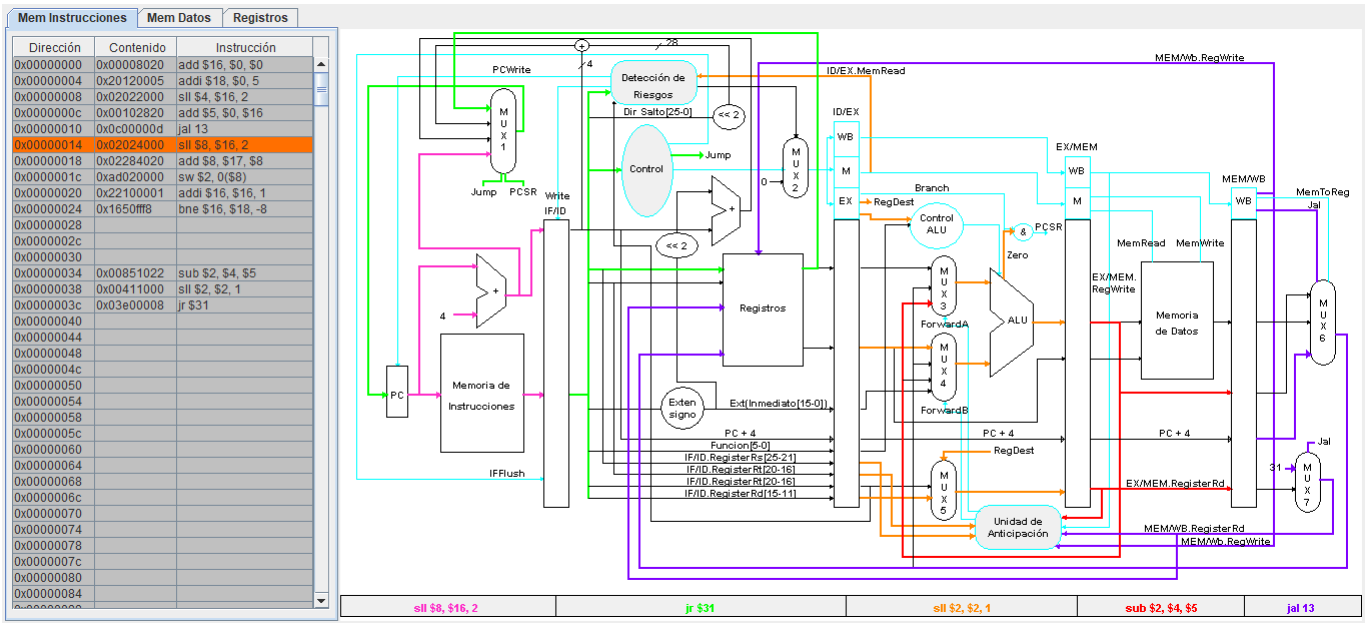


Fig. 6. Example of the behaviour of jal and jr in a program execution in UCOMIPSIM

were taken into account and included in the final version, as in the previous section it has been described.

V. Future Work

Among the improvements reserved for future versions of the application, there are some modifications that are priorities, such as those described below:

1. More detailed information of the components. Specifically, to show the circuits of some components, such as the *forwarding* and the *hazard detection* units, and how it is activated according to the progress of the program. Furthermore, to include summarized theoretical information of this component in order to reinforce the learning.
2. To include a small assembler, so that labels can be used, as well as the reordering of instructions that minimize the development of hazards.
3. To include, in addition to the pipelined computer, the option of simulating a non pipelined single-cycle MIPS computer.

VI. CONCLUSIONS

The learning of the pipeline of a datapath is fundamental for a computer engineer. When new instructions are added, as well as the *forwarding* and *hazard detection* units, the learning becomes more difficult. In this context, UCOMIPSIM simulator is used to easily show the execution of instructions on the MIPS pipelined computer. With this tool, students can analyze in detail what happens at each stage and in each component, especially when data and control hazards happen.

References

[1] G.A. Contreras, R. García, and M.S. Ramírez, "Uso de simuladores como recurso digital para la transferencia del conocimiento", Revista de Innovación Educativa, vol.2, no.1, pp.86-100, 2010.

[2] B. Nikolic, Z. Radivojevi, J. Djordjevi, and V. Milutinovic, "A survey and evaluation of simulators suitable for teaching courses in computer architecture and organization", IEEE Transactions on Education, vol.52, no.4, 2009.

[3] C. Yehezke, W. Yurzik, M. Pearson, and D. Armstrong, "Three simulator tools for teaching computer architecture: Easy CPU, Little Man Computer, and RTLSim", ACM Journal Educational Resources in Computing, vol.1, no.4, pp. 60-80, 2001.

[4] I. Aguilar, and J.R. Heredia, "Simuladores y laboratorios virtuales para Ingeniería en Computación", 2º Congreso Virtual sobre Tecnología, Educación y Sociedad (CTES'2013), Mexico, 2013.

[5] M.D. Grossi, E. Jiménez-Rey, A. Servetto, and G. Perichinsky, "Un simulador de una máquina computadora como herramienta para la enseñanza de la arquitectura de computadoras", I Jornadas de Educación en Informática y TICs en Argentina, 2005.

[6] E. Herruzo, J.I. Benavides, E. Saez, M.A. Montijano, and J.M. Paloamres, "Desarrollo de simuladores de Arquitectura de Computadores y su aplicación en la enseñanza", Congreso de Tecnologías Aplicadas a la Enseñanza de la Electrónica (TAEE'2002), Las Palmas de Gran Canaria, 2002

[7] MARS (MIPS Assembler and Runtime Simulator). Accessed: Jan. 13, 2018. [Online]. Available: http://courses.missouristate.edu/KenVollmar/mars

[8] K. Vollmar, and P. Sanderson, "MARS: An education-oriented MIPS assembly language simulator". In SIGCSE'06, pp. 239–243, ACM Press, NY, USA, 2006.

[9] H. Grünchbacher, and M. Khosravipour, "WinDLX and MIPSim pipeline simulators for teaching computer architecture", in Proc. of 1996 IEEE Symposium and Workshop on Engineering of Computer.Based Systems (ECBS'1996), pp. 412-417, 1996.

[10] J. Gómez-Luna, A. Palacios, E. Herruzo, and J.I. Benavides, "UCO.MIPSIM: Pipelined computer simulator for teaching purposes", in Proc. of VIII Congress on Technologies Applied to Electronics Teaching (TAEE'2008), pp. 1-10, 2008.

[11] I. Branovic, R. Giorgi, and E. Martinelli, "WebMIPS: A new web-based MIPS simulation environment for computer architecture education", in Proc. of 2004 Workshop on Computer Architecture Education (WCAE'2004) with the 31st International Symposium on Computer Architecture, Munich, Germany, 2004.

[12] M. Brorsson, "MipsIt - A simulation and development environment using animation for computer architecture education", in Proc. of 2002 Workshop on Computer Architecture Education (WCAE'2002) with the, 29th International Symposium on Computer Architecture, Anchorage AK, USA, 2002.

[13] K. Vollmar, and P. Sanderson, "A MIPS assembly language simulator designed for education". The Journal of Computing Sciences in Colleges, vol. 21, no. 1, 2005.

[14] D.A. Patterson, and J.L. Hennessy, "Computer organization and design MIPS edition: The hardware software interface", 5th ed., Morgan Kaufmann, 2013.