WebMIPS: A New Web-Based MIPS Simulation Environment for Computer Architecture Education

Irina Branovic, Roberto Giorgi, Enrico Martinelli
University of Siena, Italy
{branovic,giorgi,enrico}@dii.unisi.it

Abstract

We have implemented a MIPS simulation environment called WebMIPS. Our simulator is accessible from the Web and has been successfully used in introductory computer architecture course at Faculty of Information Engineering in Siena, Italy. The advantages of the Web approach are immediate access to the simulator, without installation, and a possible centralized monitoring of students' activity. WebMIPS is capable of uploading and assembling the MIPS code provided by user, simulating a five-stage pipeline step by step or completely, and displaying the values of all registers, input and output data of all pipeline elements.

1. Introduction

The study of computer architecture is a challenging field because of the high complexity involved in any computer system. To ease this complexity, different tools have been developed allowing architectures to be simulated and modified. This approach is beneficial to students approaching computer architecture for the first time, because it allows them to see the execution of actual assembly programs in the architecture. One important step is that the student makes use of simulation tools to understand concepts otherwise difficult to comprehend. Our experience, started with JCachesim cache simulator [1], indicates that Web-based lab exercising is effective, sometimes even more interesting than traditional teaching to our students. We are not alone in trying to make computer architecture education more interesting to students, as can be seen in [5], where the authors used animation for this purpose.

An extensive survey of computer architecture simulators is given in [8]. For computer architecture education, especially interesting is the category of intermediate-level simulators, targeted at students that have some background in computer architecture and need a simulator that covers the principles in more detail, but are not ready for the simulator that captures all the features of the current state-of-the-art in computer research. The simulators in this category attempt to illustrate and teach two general principles: the instruction set architecture and the micro-architecture.

In many universities, MIPS architecture is studied because it is a RISC architecture that makes understanding abstract concepts of computer design easier. Another advantage of MIPS ISA is that it is used in textbooks [2], [3], which represent a reference material for teaching computer architecture in many universities, which is also the case for our faculty. There are three widely used MIPS architecture simulators: SPIM, WinDLX and MIPSim.

SPIM [4] is an assembly language simulator for the MIPS (R2000/R3000) processor that has both a simple terminal interface and a visual, window-based interface. It implements almost entire MIPS assembler-extended instruction set (detailed SPIM description can be found in [3] with more documentation available online [4]). SPIM was extensively used in our teaching, however it lacks pipeline modeling.

WinDLX [9] and MIPSim [10] are pipeline simulators developed at the Vienna Institute of Technology and were described by authors in [11]. WinDLX models the pipeline of the MIPS-like DLX architecture described in [2]. It allows for displaying and modifying all of the information relevant to the CPU (pipeline, registers, I/O, memory), enabling/disabling pipeline forwarding, changing memory size. MIPSim [10] models the MIPS architecture as in [3], with the possibility of changing memory content, but without hazard detection and forwarding units in the pipeline.

We have decided to make a five stage MIPS pipeline simulator capable of displaying the status of almost all hardware units (more than 25) in the MIPS pipeline model, as well as hazard detection and forwarding in the pipeline. Instead of improving MIPSim, which also would have been a valid alternative for our goal, we decided to create a completely new simulator that can be executed from the Web browser window. Our simulator, called WebMIPS, eases the process of learning assembly coding, mastering pipeline, control, and datapath design. However, its major advantage is the immediate accessibility to students, without any prior installing, and the possibility of monitoring their activity over the Web.

The name WebMIPS indicates that the simulator is designed for Web use, and indeed it is written in ASP language [7] and can be started by opening a simulator Web page [6]. Another advantage of the Web based service is that the user are not required to have any special operating system for accessing this software.

WebMIPS does not support the complete MIPS instruction set; the user that wants to write assembly programs on its own must consult the list of supported instructions in order to simulate the code. Since our intention was not implementing a whole assembler, the simulator supports only the basic set of instructions, which were studied during the introductory computer architecture course.

The user can load (copy/paste) MIPS assembly file or use one of the “load-and-play” (built-in) assembly examples to follow its execution in simulator. WebMIPS is not a real assembler; however, it is able to recognize if there are errors in the provided code, and to display the line with the error. The simulator is also able of displaying the program execution step-by-step or all at once. In step-by-step mode the user can follow advancing of instructions in each stage of the pipeline, and by clicking on the constituting elements of the pipeline can see the corresponding values, input and output signals in every clock cycle. WebMIPS has forwarding always enabled, resolves pipeline hazards and displays the contents of hazard detection and forwarding units in the pipeline.

2. Detailed description of WebMIPS simulator

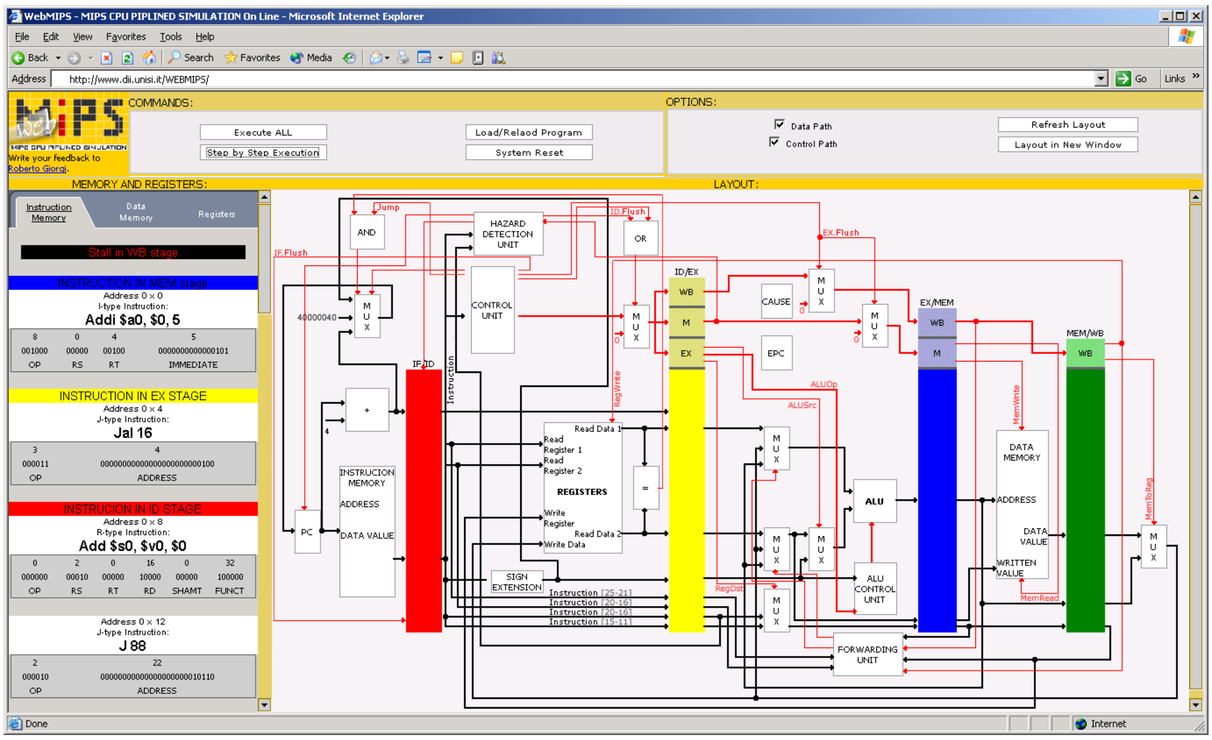
2.1 General structure

WebMIPS is a Web application and it is executed on remote servers in multiuser mode (users can execute different code at the same time). To avoid blocking of the system in case of infinite loops, erroneous references to memory and other common programming errors, we limited the execution of each uploaded program to 1000 clock cycles. On the server, all simulation parameters can be configured.

When trying to execute unsupported assembly instructions, an error is displayed and WebMIPS indicates the corresponding line number. In standard assembly language the use of directives .text and .globl is allowed, and in this case the first instruction to be executed corresponds to the .globl label. The end of execution is not specified by syscall 10, instead in WebMIPS the execution stops at the last code line.

2.2 Loading of the code

To offer the possibility of loading proprietary code to the users, we made an ASP page section, where it is possible to program in MIPS assembly and to verify whether the code is correct. By clicking on the button “Load/Reload Program” in the upper part of the WebMIPS browser window (Figure 1), the MIPS assembler is activated.



Stall in WB stage

INSTRUCTION IN MEM stage
Address 0 x 0
I-type Instruction:
Addi $a0, $0, 5

8	0	4	5
001000	00000	00100	0000000000000101
OP	RS	RT	IMMEDIATE

INSTRUCTION IN EX STAGE
Address 0 x 4
J-type Instruction:
Jal 16

3	4
000011	00000000000000000000000100
OP	ADDRESS

INSTRUCION IN ID STAGE
Address 0 x 8
R-type Instruction:
Add $s0, $v0, $0

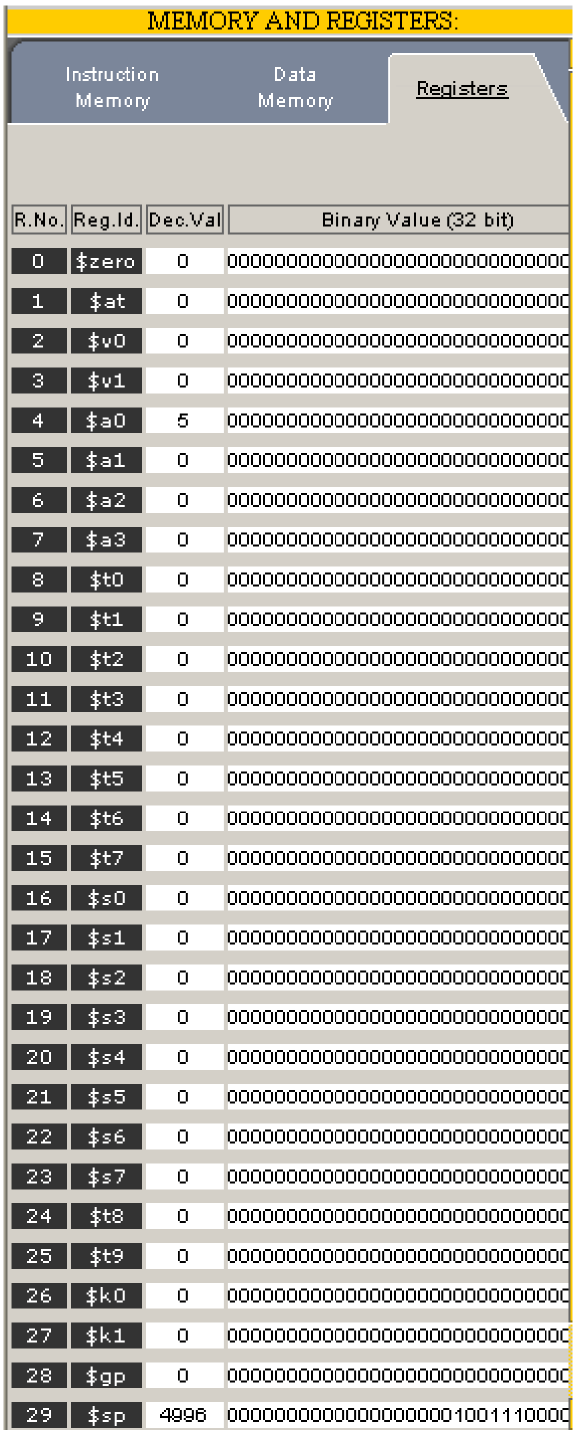
0	2	0	16	0	32
000000	00010	00000	10000	00000	100000
OP	RS	RT	RD	SHAMT	FUNCT

Address 0 x 12
J-type Instruction:
J 88

2	22
000010	00000000000000000000010110
OP	ADDRESS

Figure 1. WebMIPS window during execution. Note the Load/Reload Program button in the upper part. The wires can be hidden to have an easier reading of the CPU units.

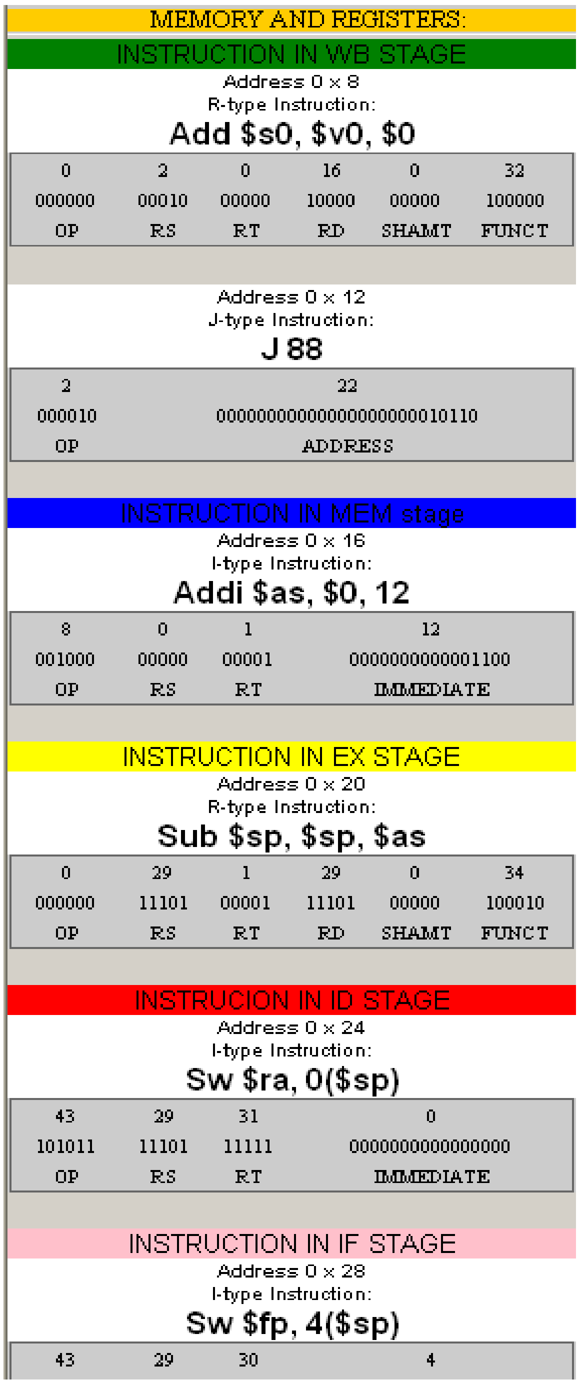
Not all options of the real assembly were implemented, since our goal was to demonstrate the execution of base instructions explained during our computer architecture course. However, almost all MIPS instructions can be written by combining the implemented instructions. We included a set of simple assembly programs with the scope of demonstrating its execution in MIPS pipeline.


MEMORY AND REGISTERS:

R.No.	Reg.Id.	Dec.Val	Binary Value (32 bit)
0	$zero	0	00000000000000000000000000000
1	$at	0	00000000000000000000000000000
2	$v0	0	00000000000000000000000000000
3	$v1	0	00000000000000000000000000000
4	$a0	5	00000000000000000000000000000
5	$a1	0	00000000000000000000000000000
6	$a2	0	00000000000000000000000000000
7	$a3	0	00000000000000000000000000000
8	$t0	0	00000000000000000000000000000
9	$t1	0	00000000000000000000000000000
10	$t2	0	00000000000000000000000000000
11	$t3	0	00000000000000000000000000000
12	$t4	0	00000000000000000000000000000
13	$t5	0	00000000000000000000000000000
14	$t6	0	00000000000000000000000000000
15	$t7	0	00000000000000000000000000000
16	$s0	0	00000000000000000000000000000
17	$s1	0	00000000000000000000000000000
18	$s2	0	00000000000000000000000000000
19	$s3	0	00000000000000000000000000000
20	$s4	0	00000000000000000000000000000
21	$s5	0	00000000000000000000000000000
22	$s6	0	00000000000000000000000000000
23	$s7	0	00000000000000000000000000000
24	$t8	0	00000000000000000000000000000
25	$t9	0	00000000000000000000000000000
26	$k0	0	00000000000000000000000000000
27	$k1	0	00000000000000000000000000000
28	$gp	0	00000000000000000000000000000
29	$sp	4996	00000000000000000001001110000

Figure 2: Registers during execution.

The functioning of our simulator can be easily understood by using some of the simple built-in (called "load-and-play") programs. The simulator keeps track of the code in execution and it can be easily modified in any moment by clicking on the "Load/Reload Program" button.


MEMORY AND REGISTERS:

INSTRUCTION IN WB STAGE

Address 0 x 8
R-type Instruction:
Add $s0, $v0, $0

OP	RS	RT	RD	SHAMT	FUNCT
0	2	0	16	0	32
000000	00010	00000	10000	00000	100000

Address 0 x 12
J-type Instruction:
J 88

OP	ADDRESS
2	22
000010	00000000000000000000010110

INSTRUCTION IN MEM stage

Address 0 x 16
I-type Instruction:
Addi $as, $0, 12

OP	RS	RT	IMMEDIATE
8	0	1	12
001000	00000	00001	0000000000001100

INSTRUCTION IN EX STAGE

Address 0 x 20
R-type Instruction:
Sub $sp, $sp, $as

OP	RS	RT	RD	SHAMT	FUNCT
0	29	1	29	0	34
000000	11101	00001	11101	00000	100010

INSTRUCION IN ID STAGE

Address 0 x 24
I-type Instruction:
Sw $ra, 0($sp)

OP	RS	RT	IMMEDIATE
43	29	31	0
101011	11101	11111	0000000000000000

INSTRUCTION IN IF STAGE

Address 0 x 28
I-type Instruction:
Sw $fp, 4($sp)

43	29	30	4

Figure 3: Instruction memory in the middle of execution.

2.2 Program execution

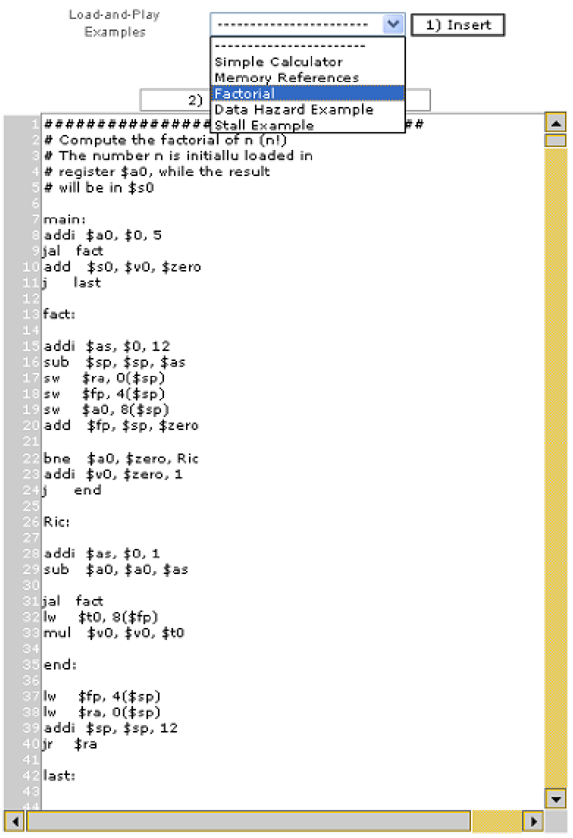
In order to allow users to follow program execution, the left part of the browser window is dedicated to information regarding register file, data and instruction memory. Instruction memory displays the mnemonic, memory address, type, binary translation, symbolic representation, field values and current position in the pipeline for every instruction in execution (Figures 2, 3).
The page displaying data memory can visualize single words, a word interval, or the whole memory contents. The register page demonstrates the binary content of 32 MIPS registers, which can be identified either by register number or their symbolic identifier.

The central part of the browser page is dedicated to five-stage MIPS pipeline. Since the major scope of the simulator was to facilitate understanding of pipeline principles, a user can click on any desired element of the pipeline (for example, ALU, hazard detection unit, or even a simple multiplexer) to show its input and output data. A good feature of WebMIPS is the possibility of tracking every instruction in each pipeline element by simply looking at the central graphical screen. Additionally, displaying of control/data wires can be turned on/off using a corresponding check-box.

Once loaded, a program can be executed in two modes: step-by-step or completely. In step-by-step mode, after each clicking of the "Step-by-Step Execution" the pipeline stages are updated and the user can see the changes in memory and detailed pipeline logic. After the execution has completed, the total number of clock cycles is calculated and displayed in the left-hand menu. Complete execution of the program should be is used only for verifying the correctness of the assembly code.

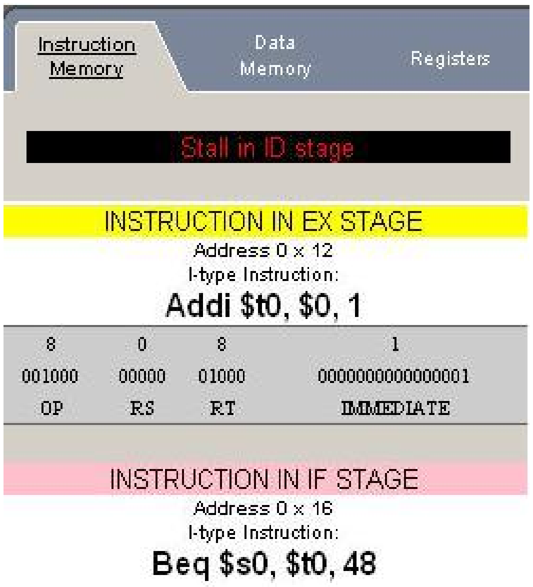
2.3 Analyzing pipeline data hazard and forwarding

In the implementation, branch decision is in the Decode stage of the pipeline to save one cycle. Data hazards created in this way are detected in hazard detection unit, and resolved via forwarding unit, which are shown in graphic representation of the pipeline. Among "load-and-play" programs user can find a simple four-operation calculator; the loading of this example is shown in Figure 4. We will use this simple example to illustrate the functioning of hazard detection and forwarding in the pipeline. The top of the left-hand menu lists the pipeline stages in stall during the execution of the program (Figure 5).



```
################                    ##
# Compute the factorial of n (n!)
# The number n is initiallu loaded in
# register $a0, while the result
# will be in $s0

main:
addi  $a0, $0, 5
jal   fact
add   $s0, $v0, $zero
j     last

fact:

addi  $as, $0, 12
sub   $sp, $sp, $as
sw    $ra, 0($sp)
sw    $fp, 4($sp)
sw    $a0, 8($sp)
add   $fp, $sp, $zero

bne   $a0, $zero, Ric
addi  $v0, $zero, 1
j     end

Ric:

addi  $as, $0, 1
sub   $a0, $a0, $as

jal   fact
lw    $t0, 8($fp)
mul   $v0, $v0, $t0

end:

lw    $fp, 4($sp)
lw    $ra, 0($sp)
addi  $sp, $sp, 12
jr    $ra

last:
```

Figure 4: A simple calculator program (among built-in examples) loaded.



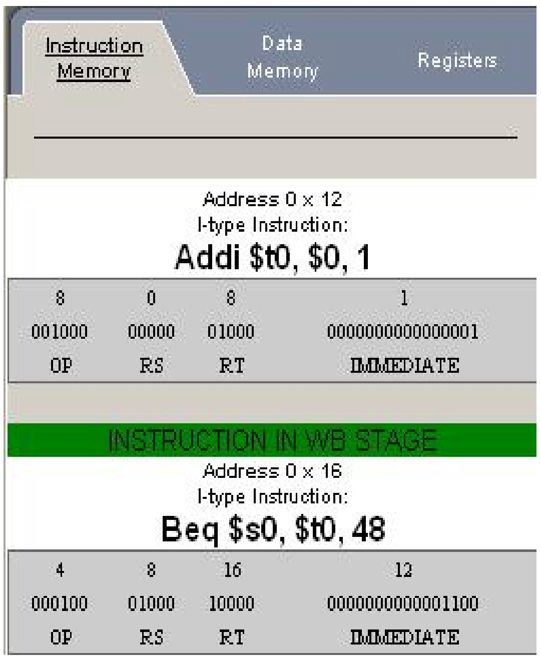
Stall in ID stage

INSTRUCTION IN EX STAGE
Address 0 x 12
I-type Instruction:
Addi $t0, $0, 1

8	0	8	1
001000	00000	01000	0000000000000001
OP	RS	RT	IMMEDIATE

INSTRUCTION IN IF STAGE
Address 0 x 16
I-type Instruction:
Beq $s0, $t0, 48

Figure 5: The top of the memory window displays a stage in stall.



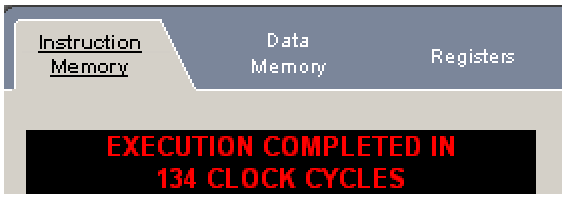
Address 0 x 12
I-type Instruction:
Addi $t0, $0, 1

8	0	8	1
001000	00000	01000	0000000000000001
OP	RS	RT	IMMEDIATE

INSTRUCTION IN WB STAGE
Address 0 x 16
I-type Instruction:
Beq $s0, $t0, 48

4	8	16	12
000100	01000	10000	0000000000001100
OP	RS	RT	IMMEDIATE

Figure 6: Stall passed through the pipeline.



EXECUTION COMPLETED IN
134 CLOCK CYCLES

Figure 7: When the execution finishes, the top of the memory window shows the total number of clock cycles.

The pipeline with data hazard resolved is displayed on Figure 6. When the execution of the program finishes, the simulator displays total number of cycles (Figure 7). By clicking on the hazard detection and forwarding units in the pipeline a user can see the corresponding signals and follow the propagation of the stall through the pipeline (Figure 8).

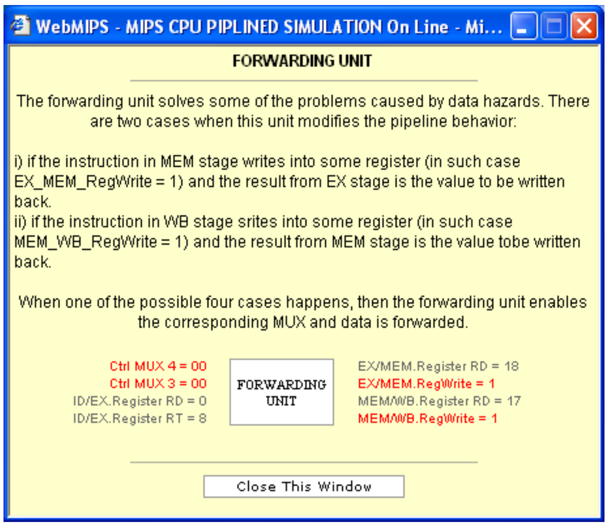
Conclusions

We have implemented a Web-based MIPS pipeline simulator called WebMIPS. Our simulator is publicly accessible and it displays execution in the Web browser window, and is capable of detecting and resolving hazards in the pipeline. The WebMIPS software was used in introductory computer architecture course at University of Siena, Italy as an auxiliary resource for explaining pipeline principles.

We received a good feedback from our students, who also appreciated its availability from any client computer (independently from the installed operating system), the possibility of executing on any Internet-enabled PC without prior installation, and its ease of use. Further plans for WebMIPS development include extending the supported instruction set to include all MIPS (R2000/R3000) instructions.

Acknowledgements

We are particularly grateful to the students: Mirko Casini, Riccardo Donati, Alem Gracic, Luca Peruzzi for the initial implementation and the testing of WebMips.



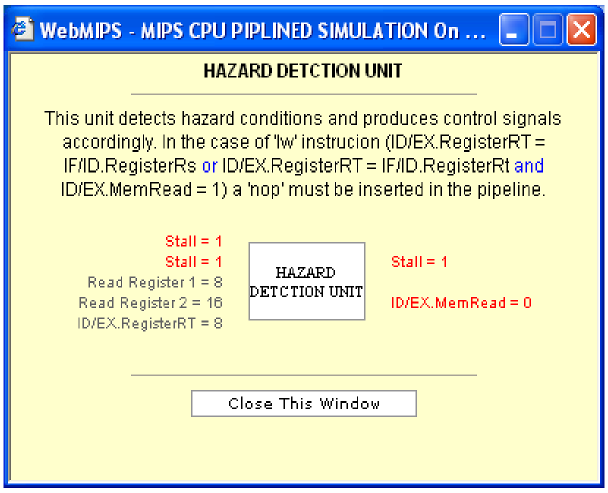
FORWARDING UNIT

The forwarding unit solves some of the problems caused by data hazards. There are two cases when this unit modifies the pipeline behavior:

i) if the instruction in MEM stage writes into some register (in such case EX_MEM_RegWrite = 1) and the result from EX stage is the value to be written back.
ii) if the instruction in WB stage srites into some register (in such case MEM_WB_RegWrite = 1) and the result from MEM stage is the value tobe written back.

When one of the possible four cases happens, then the forwarding unit enables the corresponding MUX and data is forwarded.

Ctrl MUX 4 = 00
Ctrl MUX 3 = 00
ID/EX.Register RD = 0
ID/EX.Register RT = 8
EX/MEM.Register RD = 18
EX/MEM.RegWrite = 1
MEM/WB.Register RD = 17
MEM/WB.RegWrite = 1



HAZARD DETCTION UNIT

This unit detects hazard conditions and produces control signals accordingly. In the case of 'lw' instrucion (ID/EX.RegisterRT = IF/ID.RegisterRs or ID/EX.RegisterRT = IF/ID.RegisterRt and ID/EX.MemRead = 1) a 'nop' must be inserted in the pipeline.

Stall = 1
Stall = 1
Read Register 1 = 8
Read Register 2 = 16
ID/EX.RegisterRT = 8
Stall = 1
ID/EX.MemRead = 0

Figure 8: Details on forwarding and hazard detection in the pipeline can be seen by clicking on the corresponding unit.

References

[1] I. Branovic, R. Giorgi, A. Prete, Web-based training on computer architecture: The case of JCachesim, *Proceedings of the Workshop on Computer Architecture Education*, pp. 56-60, May 2002, Anchorage, Alaska.

[2] J. L. Hennessy and D. A. Patterson, *Computer Architecture – A Quantitative Approach, 3rd edition*, Morgan Kaufmann Publishers, 2002.

[3] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface,* 2nd edition, Morgan Kaufmann Publishers, 1997.

[4] SPIM simulator Home Page, http://cs.wisc.edu/~larus/spim.html

[5] M. Brorsson, MipsIt - A Simulation and Development Environment Using Animation for Computer Architecture Education, *Proceedings of the Workshop on Computer Architecture Education*, pp. 65-72, May 2002. Anchorage, Alaska.

[6] WebMIPS Home Page, http://www.dii.unisi.it/~giorgi/WEBMIPS/

[7] C. Payne, *Teach Yourself ASP.NET in 21 days,* 2nd edition, SAMS, 2003.

[8] W. Yurcik, G. S. Wolffe, M. A. Holiday, A Survey of Simulators used in Computer Organization/Architecture Courses, *Proceedings of Summer Conference on Computer Simulation*, pp. 524-529, Orlando, Florida, July 2001.

[9] WinDLX Simulator Download Page, ftp://ftp.mkp.com/pub/dlx/

[10] MIPSim Simulator Download Page, http://mouse.vlsivie.tuwien.ac.at/lehre/rechnerarchitekturen/download/Simulatoren/

[11] H. Grünbacher, M. Khosravipour, WinDLX and MIPSim Pipeline Simulators for Teaching Computer Architecture, *Proceedings of IEEE Symposium and Workshop on Engineering of Computer Based Systems*, pp. 412-417, Friedrichshafen. Germany, March 1996.