Table of Contents:

+

About MIPSim

MIPSim is intended to be a supplement to the book:

Computer Organization an Design
The Hardware/Software Interface

by **John L. Hennessy** and **David A. Patterson**.

While this book is an excellent introduction to the basics of computer architecture, experience shows that many students, especially those coming from less hardware-oriented studies, encounter difficulties in understanding the consequences and even concepts of pipelining. Presenting the dynamic process of instructions floating through a pipeline using cycle diagrams

often appears somewhat clumsy to students with little or no previous knowledge of computer architecture and digital logic design.

MIPSim was designed to offer a more feasible way of demonstrating a pipelined architecture. It simulates and visualizes what happens within a pipeline similar to the pipelines presented in the above mentioned book and thus may hopefully aid its readers in understanding the basics of pipelining.
The user can write small programs (although currently there is only a subset of the MIPS instruction set implemented) and watch the pipeline doing its work, modify the program and the content of data memory and register file   on the fly   and go on simulating to see the effects.

Vienna University of Technology
Institut fÃ¼r Technische Informatik
Prof. Herbert GrÃ¼nbacher

TreitlstraÃ e 3/182-2
A-1040 Vienna/Austria

Tel.: +43 (1) 58801 - 8150 Fax.: +43 (1) 5869697
email: hg@vlsivie.tuwien.ac.at
http://www.vlsivie.tuwien.ac.at/welcome.html

+

Getting started

If you should encounter problems in running this program, please see chapter 9: Troubleshooting.

For a quick test of the programs functionality, there is an example program to demonstrate the simulator. To load it:

1.   select FILES from the main menu
2.   select OPEN from the drop-down menu
3.   in the file-listbox, double-click on *demo.mp*
      or
      select it and click the OK-button.

The listbox at the left of the schematic should contain some opcodes.

To start the simulation, first drag the opcode-arrow at the arrow on its right side onto the first opcode (in case it isn t already there) to instruct MIPSim it should start with this one (this symbol indicates the line at which the opcodes leave the listbox).

Now click on the Simulation-Start button or select SIMULATION from the main menu and then START from the drop-down menu.

You should now see the instructions   floating   through the pipeline, i.e. being shifted through the top line of the schematic. Each instruction entering the pipeline has a certain color assigned, and the buses and connections are continually redrawn in the same color to reflect the   flow   of the corresponding signals.

If the pipeline works, you can start with the <u>tutorial</u>.. In case of troubles see chapter 9: <u>Troubleshooting</u>.

+

Tutorial

This tutorial provides a brief introduction to simulating with MIPSim.

When you start MIPSim, <u>the main window</u> pops up.
You see the the schematic of the processor with all buses in grey, indicating that their state is undefined as no instructions are in the pipeline.

There are three special symbols within the schematic:
The instruction memory (labeled *instr.*), the register file (*register*), and the data memory (*data*).
Clicking on one of these symbols opens a dialog which lets you modify its contents.

To enter a small program, click on the instruction memory symbol within the schematic.

<u>The assembler window</u> appears.
After clicking on a field within the opcode-column, it receives the caret and you can edit its contents.
Now click on the topmost field in the opcode-column and type:

LW $1,0($0)

then press <ENTER>.
MIPSim performs a syntax-check, and if the input is correct, the caret is set to the next line (within the opcode column).

Now try an incorrect input, for example:

ADD $1   (correct opcode, but wrong parameter format)

On<ENTER>, a window pops up, stating that the input was syntactically incorrect and, as MIPSim recognizes the ADD opcode, displaying the expected format.

In case of an incorrect input, the caret remains within the current field and the text is selected for overwrite to let you correct its contents.
If you clear the input with the <DEL> key and press <ENTER>, MIPSim reverts to the instruction that was previously entered for this memory location. If you enter a single blank, MIPSim replaces the instruction with a NOP.

Now please enter the following lines immediately below the LW $1,0($0):

NOP
NOP
NOP
ADD $3,$1,$2
SUB $4,$1,$2
OR $5,$1,$2

3

Click on the close-button, and the assembler dialog disappears.

Back in the main window, you should see the previously entered assembler statements listed in the listbox to the left of the schematic (maybe you need to scroll up to see the top line).

This listbox also contains the opcode arrow , which indicates the line where the opcodes leave the listbox to enter the pipeline. Make sure it is positioned on the first line (the LW-instruction); if necessary, drag it at its right edge.

The short example program loads a value from the data memory into register $1 and then performs some operations based on the values in registers $1 and $2. Therefore we need to enter the initial values for those registers.

The value for register $1 is loaded from the data memory, so we have to put a value into the appropriate memory location.

The contents of the data memory can be examined and modified in a similar way as with the instruction memory, namely by clicking on the data memory symbol (labeled *data*).

The data window pops up:

The listbox shows the data memory s contents. You may modify any of the words, just click on it and edit the text. The numbers are always given in hexadecimal format.
So, click on the leftmost value within the first line, which is the data word at address 0, and enter 5. Then click on the close-button, and the data dialog should disappear.

The value for register $2 must be entered directly into the register. To do so, click on the register file.

The register window pops up:

You see editable fields for each register (except $0, which is hardwired to 0).
Click on the value of $2 and enter 3 and hit <RETURN> (again, values are given in hexadecimal).

Now, we have the initial values for the two registers we use. You could close the register window now (by clicking on the close-button in the upper right corner, by clicking on the register button in the toolbar, or by clicking on the register file symbol within the schematic. However, to provide the opportunity of watching the registers contents change on running a program, you can also keep the register dialog open. Just resize it so that the registers you re interested in are visible and move the dialog to an appropriate place.

Make sure the opcode arrow is positioned on the first line (the LW-instruction); if necessary, drag it at its right edge.
Now we are ready to start the simulation.

So, click on the play-button . The first instruction should immediately shift into the pipeline, and you should see the signals on the buses in the same color as the opcode appearing in the line above the schematic. In play mode, you can watch the signals propagate on the buses. Slow mode and fast mode offer different signal propagation speeds. However, if you just want to see the results of each step, you can use the step-button .

When the instructions reach the write-back stage, the register contents should be updated accordingly in the register dialog.

Wait until the last instruction has shifted through the pipeline (only NOP s left in the top line), then stop the pipeline by clicking on the stop button . The listbox scrolls back up to the first opcode and all buses now appear in grey indicating an undefined state.

The results should be in the registers $3,$4, and $5.

We have watched the signals shift through the pipeline stages and got the expected results in the registers.

Now, let s take a closer look at the pipeline s work.

Make sure the program is reset and the opcode arrow is located at the first instruction (LW).
Now click once onto the step-button within the toolbar .

The LW-instruction enters the instruction fetch (IF) stage of the pipeline.
Click repeatedly on the step-button until the ADD-instruction is in the execution stage (EX).

MIPSim enables you to inspect the current state of each bus. Click (with the left mouse button), for example, once on each of the two ALU-inputs and once on the ALU-output.
For each of the buses, a label pops up, displaying the current state of the bus in hexadecimal, and the value of the output bus should be the sum of the values of the input buses.
Likewise, you can inspect any other bus within the schematic.

Now click once again on the step-button. The labels are updated automatically when the state of the bus changes. So, when simulating, you can activate the labels on the buses you are interested in and MIPSim updates the values as the instructions shift along.

To hide a label, just click once on it, and it disappears.
Alternatively, by clicking on the clear labels button , all labels can be removed.

In this example, we have three NOPs after the load instruction. Their purpose is to delay the execution of the commands referencing the loaded value until the registers are updated accordingly. Now lets see what would happen without them:
Mark the ADD, SUB and OR instructions, i.e. click the left mouse button on the line containing the ADD statement (preferably within the address column) and move the mouse to the line with the OR statement, then release the mouse button. These three lines should appear in red. Now, drag them up right behind the LW instruction: click the *right* mouse button on one of the marked instructions and move the mouse up. MIPSim draws an insertion mark where the lines will be inserted upon releasing the mouse button. So, move the mouse up until the insertion mark is right behind the LW instructions and then release the mouse button.

Now, clear the values in the registers $3-$5 to zero, either by directly entering the zeroes or by clicking the clear button in the register dialog and reentering the value 3 for register $2.

You may want to inspect the write inputs and the outputs of the register file,so click on them once to get the labels.

Start the simulation again, and you will see the processor calculating nonsense, because the result from the LW instruction is not written to the register file on time.

Alternatively, instead of moving the NOPs, you can use the opcode-arrow to modify the order of execution: To do so, simulate in single step mode using the step-button , and right after the first step drag the opcode-arrow from the first NOP directly onto the ADD-instruction, thus omitting the NOPs from execution.

With the presented features, you can write your own programs and examine what the pipeline does. There are also some more examples (demo1 - demo3) you may want to try out.

The contents of the instruction and data memories as well as the content of the register file may be saved/loaded separately by clicking the save/load buttons. LOAD/SAVE from the FILE menu loads/saves the contents of all three of them.

+
The main window

The main window ist mostly occupied by the processor schematic. It shows the five stages (labeled right above) of the processor, each reduced to the most relevant architectural units : the PC, the instruction an data memory, the ALU, the main control unit, the register file each separated by latches.

 The listbox to the left of the schematic shows the content of the instruction memory as an assembler program. The listbox scrolls up as the simulation goes on, and the instructions   move   into the pipeline, each in the corresponding color.
The arrow-symbol marks the line where the opcodes   leave   the listbox. By dragging it at the right edge it can be moved up and down the listbox, thus making it possible to determine how many of the previous and following lines of code should be visible. It can also be used to select the next opcode to enter the pipeline. Note: The opcode arrow does not scroll along with the listbox content. So, upon scrolling the listbox, you may have to adjust its position.

Above the schematic, right below the caption of the main window, there is the toolbar:

It gives quick access to the most common commands, namely (from left to right):

 Open the assembler window.
 Open the register window.
 Open the data memory window.

 Remove all labels from the schematic.

 Start the simulation.
 Pause the simulation.
 Stop the simulation.
 Simulate fast.
 Simulate slow.
 Perform a single step.

 Print the current schematic state (including labels).

Display the about dialog.
Display this online help.

+
#The assembler window

Within the assembler window, you can enter and modify the contents of the program memory.
It can be opened in one of three ways:

1. Via the main menu by selecting INSTRUCTION MEMORY/VIEW.
2. By clicking on the assembler button in the toolbar
3. Or by clicking on the symbol labeled instruction memory within the schematic.

The listbox on the left contains the actual program. It consists of four columns:

address: displays the (byte) address.
label: displays a label used as a branch or jump target.
opcode: The opcode itself.
word: The opcode in hexadecimal (four bytes).

You can change the content of any field in the label, opcode and word columns by selecting it and editing the text; the address column is read only.

The labels entered in the label column may be used as the destination address in a BEQ or a BNEQ instruction. However, as this is a line-by-line assembler, each label has to be defined before it can be referenced.

When you enter an opcode in the opcode-column, MIPSim automatically performs a syntax check and notifies you if the entry is syntactically incorrect.

The listbox on the right shows the available instructions as well as their syntax (this is just informational; selecting instructions will *not* copy them to the assembler listbox).

The OK button closes the window.

With the clear all button , the instruction memory content may be reset to contain all zeroes, ie. NOPs.

The load button opens a file dialog displaying all previously saved programs in the active path (*.mp).
To load one, double-click it or select it and click the OK button.

Click if you want to save the current program to disk. In the file dialog, enter the name to be used and click ok (or press ENTER). MIPSim will ask for confirmation if the given file already exists.

+
The data window

This window may be used to display and modify the contents of the data memory.
It can be opened in one of three ways:

1. Via the main menu by selecting DATA MEMORY and then VIEW from the popup menu.
2. By clicking on the data memory button in the toolbar
3. Or by clicking on the symbol labeled data memory within the schematic.

The listbox contains four columns of data words a line, each preceded by the adress of the first data word. You can edit any data word by clicking on it. Number format is hexadecimal.

The OK button closes the window.

With the clear all button , the data memory content may be reset to contain all zeroes.

The load button opens a file dialog displaying all previously saved memory files (*.md).
To load one, double-click it or select it and click the OK button.

Click if you want to save the current data memory content to disk. In the file dialog, enter the name to be used and click ok (or press ENTER).

+
The register window

This window displays the contents of the registers and lets you modify them by clicking on a value.
It can be opened in one of three ways:

1. Via the main menu by selecting REGISTERS and VIEW from the popup-menu.
2. By clicking on the registers button in the toolbar
3. Or by clicking on the symbol labeled register file within the schematic.

Click the load button to open a file dialog in which you can select a *.mr-file containing previously saved register contents.

The clear button resets all register s contents to zero.

By clicking the save button , you may enter a filename under which the current register contents are saved.

+#Printing

By selecting PRINT in the FILE menu, you can send the current state of the pipeline to the printer.
The schematic is then printed along with the opcode line above it. It is automatically scaled to fit the paper format.

+
Troubleshooting

Known problems:

Schematic:

The schematic resembles to the examples in the Hennessy/Patterson book, but there are some (although negligible) differences, e.g.: the ALU control bus consists of four bits. This could possibly cause some confusion.

Forwarding within the register file is implemented, so the register file supplies the correct value if the instruction in the ID stage

reads a register written by the instruction in the WB stage.

You cannot display more than one label per bus. Please note: A bus segment ending in a branch is not regarded as a separate bus (which could have its own label) but rather a part of one of the buses leading further.
Therefore, with the two control-lines in the WB-stage (mtr and mw), when you click up a label on the bus-segment before they branch out, you will only get a label for the mtr-signal.

Assembler:

The processing of labels is quite coarse. As MIPSim utilizes a line-by-line assembler (definitely not designed for complex programs), labels have to be defined before they can be used. On forward references, you may have to enter a dummy address and change it to the destination label later on.

Be careful when moving lines. Currently, references out of or into the moved part of the program are *not* automatically updated. You may have to enter the correct addresses or labels again.

If you want to select lines in order to move them, do not begin the selection in the edit field currently having the caret, because this mouse click will be interpreted as a caret movement within the field rather than the begin of a selection. It is recommended to select lines by clicking on them within the address column, as the address is read only and thus can never have the caret.

In case the caret is in an edit field within a selected line, this edit field will appear in black rather than red.

Problems actually running MIPSim:

MIPSim does not need an installation procedure. Just create a subdirectory, copy or unzip the files into it, and run the executable (you may want to set up an icon). However, the *ra_cpu.sch* file and the help file must be located in the same directory as the executable.

The are two separate versions of MIPSim, a 16bit version for Windows 3.1 and a 32bit version for Windows NT (which could also be used with Windows 95). Make sure to use the correct version for your system. (The 16bit version should work with NT, but in order to run the 32bit version under Windows 3.1, you will need the Win32s-extension, which is freely available from Microsoft.).

In case MIPSim does not run at all, try to unzip the appropriate version (mipsim16.zip or mipsim32.zip) again into an empty directory. After that, this directory should contain at least the following files:

1. mipsim32.exe(or mipsim16.exe for the 16bit version).
2. mipsim32.hlp (or mipsim16.hlp for the 16bit version).
3. ra_cpu.sch

There should also be some more files containing some example programs. However, only the three files named above are absolutely necessary in order to run MIPSim. If one of these files is missing, the zip-file is incomplete and you will need to download it again.

If this does not fix the problem contact:

Vienna University of Technology

Institut fÃ¼r Technische Informatik
Prof. Herbert GrÃ¼nbacher

TreitlstraÃ e 3/182-2
A-1040 Vienna/Austria

Tel.: +43 (1) 58801 - 8150 Fax.: +43 (1) 5869697
email: hg@vlsivie.tuwien.ac.at
http://www.vlsivie.tuwien.ac.at/welcome.html

+
#Deinstalling MIPSim

To deinstall MIPsim, just remove the files mipsim32.* (or mipsim16.* for the 16 bit version) and ra_cpu.sch along with any *.mp, *.md and *.mr files containing MIPSim programs. Apart from these, MIPSim itself does not leave any other files on your harddisk.

+$
#KThe schematic

MIPSim was designed to support readers of the book   Computer Organization and Design - The Hardware/Software Interface   by Hennessy/Patterson in understanding the concept of a pipeline. Therefore, the schematic utilized in MIPSim is very similar to the ones presented by Hennessy/Patterson, especially in the chapters 5 and 6.
The current version leaves forwarding and hazard detection out to keep the schematic as simple as possible.

See: chapters 5 and 6 in :
  Computer Organization and Design - The Hardware/Software Interface   Hennessy/Patterson

---

+$#Arithmetic Logic Unit (ALU)

The ALU in this simulator can perform various operations on 32-bit values. The Control Unit determines the operation to be performed via the 4-bit *Alu operation bus.*
Carry and *Overflow* are not implemented.

See: pages 182 and 278 in :
  Computer Organization and Design - The Hardware/Software Interface   Hennessy/Patterson

---

+$#Adder

This simulator contains two 32-bit adders: One in the IF stage to increment the PC, and the other in the EX stage to add the sign-extended 16-bit offset to the PC.

See: pages 175 and 277 in :
  Computer Organization and Design - The Hardware/Software Interface   Hennessy/Patterson

+$#Constant Value

This symbol shows that the corresponding input is   hard-wired .

+$#Instruction Memory (IM)

The instruction memory in the IF stage is addressed by the PC and delivers the corresponding 32-bit value to the IF/ID-Latch

See: pages 272 and 277 in :
  Computer Organization and Design - The Hardware/Software Interface   Hennessy/Patterson

+$#Data Memory (DM)

On the zero-to-one transition of the MemWrite input the 32-bit value on the data input is stored at the address on the address input.

See: pages 272 and 280 in :
  Computer Organization and Design - The Hardware/Software Interface   Hennessy/Patterson

+$#KRegister File

The register file in this simulator contains 32 32-bit registers.
Register 0 is hardwired to zero.

See: pages 272,278 and 279 in :
  Computer Organization and Design - The Hardware/Software Interface   Hennessy/Patterson

+$#And-Gate

Produces the logical   and   of all of it s inputs. In this case it combines the Zero output of the ALU and the branch bit from the Control Unit to select the appropriate PC value.

See: page 184 in :
  Computer Organization and Design - The Hardware/Software Interface   Hennessy/Patterson

+$#KSign Extension Unit

Extends the 16-bit offset value contained in the opcode to 32 bit so that it can be added to the current address, which is 32 bit.

See: page 280 in :
  Computer Organization and Design - The Hardware/Software Interface   Hennessy/Patterson

+$#Multiplexor (MUX)

Used to select an arbitrary input to be fed through to the output. In this schematic, multiplexers are used to distribute the control signals according to the respective opcode.

See: page 184 in :
  Computer Organization and Design - The Hardware/Software Interface   Hennessy/Patterson

+$#KControl Unit (CU)

From the opcode word as its input, the Control Unit generates several signals to control the operation of most of the elements in the schematic.

See: pages 291-306 in :
  Computer Organization and Design - The Hardware/Software Interface   Hennessy/Patterson

+$#Latch

The latches are used to store the signals computed within one stage until the next stage cycle begins, when they are fed through to the following stage.

See: Appendix B page 22 in :
  Computer Organization and Design - The Hardware/Software Interface   Hennessy/Patterson

+$#Program Counter (PC)

The program counter is a special latch which is used to store the current address.

See: page 277 in :
  Computer Organization and Design - The Hardware/Software Interface   Hennessy/Patterson