ProcSim MIPS v2.0

James Garton (c) 2005

This program simulates a MIPS R2000 Processor. It can display a number of datapaths through the processor, showing how each component of the architecture works with the input. Assembly programs can be input in to the program using supported instructions. These vary with the architecture used. Different architectures can be created or loaded using an XML format, in conjunction with an in program diagram designer. You can see an explanation of the XML format used below.

This program is great for learning how the processor works. And can be used to design your own architectures to practice your understanding (although you cannot add new instructions to the ISA - only the following are supported: add, sub, and, or, slt, lw, sw, beq, addi, andi, ori, j).

You need Java v1.5 to run this program.
You can download it from http://www.java.com/en/download/manual.jsp.

Minimum System Requirements (not tested!):

OS: Windows 95-XP, (Linux).
Processor: 1GHz CPU
Resolution Monitor: 1024*768

Recommended:

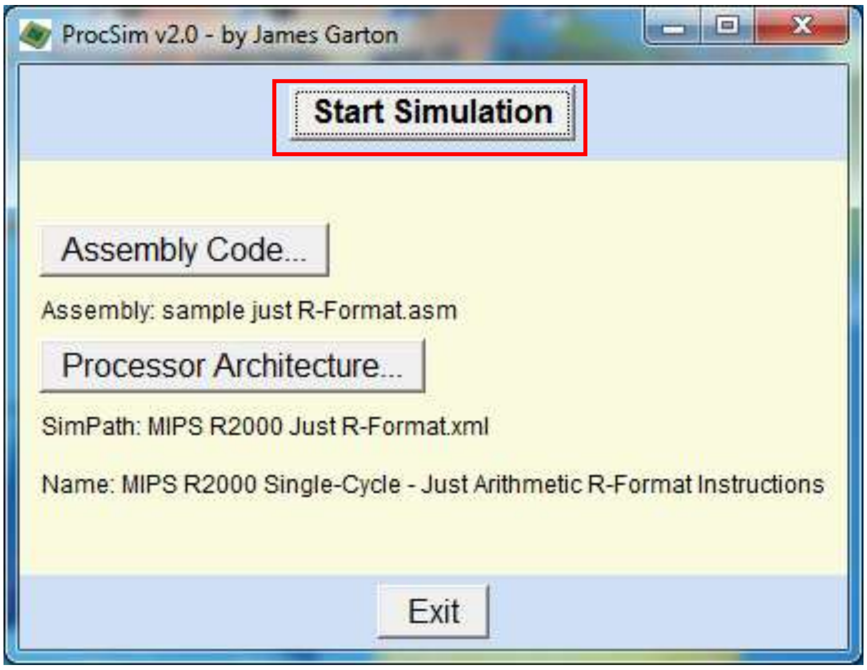
OS: Windows 2k-XP
Processor: >1.5GHz
Resolution Monitor: 1920*1080 (Full HD)

Running ProcSim MIPS

Make sure you have the Java runtime environment (JRE >= v1.5) before running ProcSim.

Windows:

Execute the provides windows script: **ProcSim.vbs**



Assembly: sample just R-Format.asm

SimPath: MIPS R2000 Just R-Format.xml

Name: MIPS R2000 Single-Cycle - Just Arithmetic R-Format Instructions

Fig. 1 - ProcSim startup

A number of architectures and assembly programs are included, and one is loaded at startup. So it is possible to just start straight away and click "Start Simulation".

1. Simulation Usage Help

1.1 Example Single-cycle Datapath:

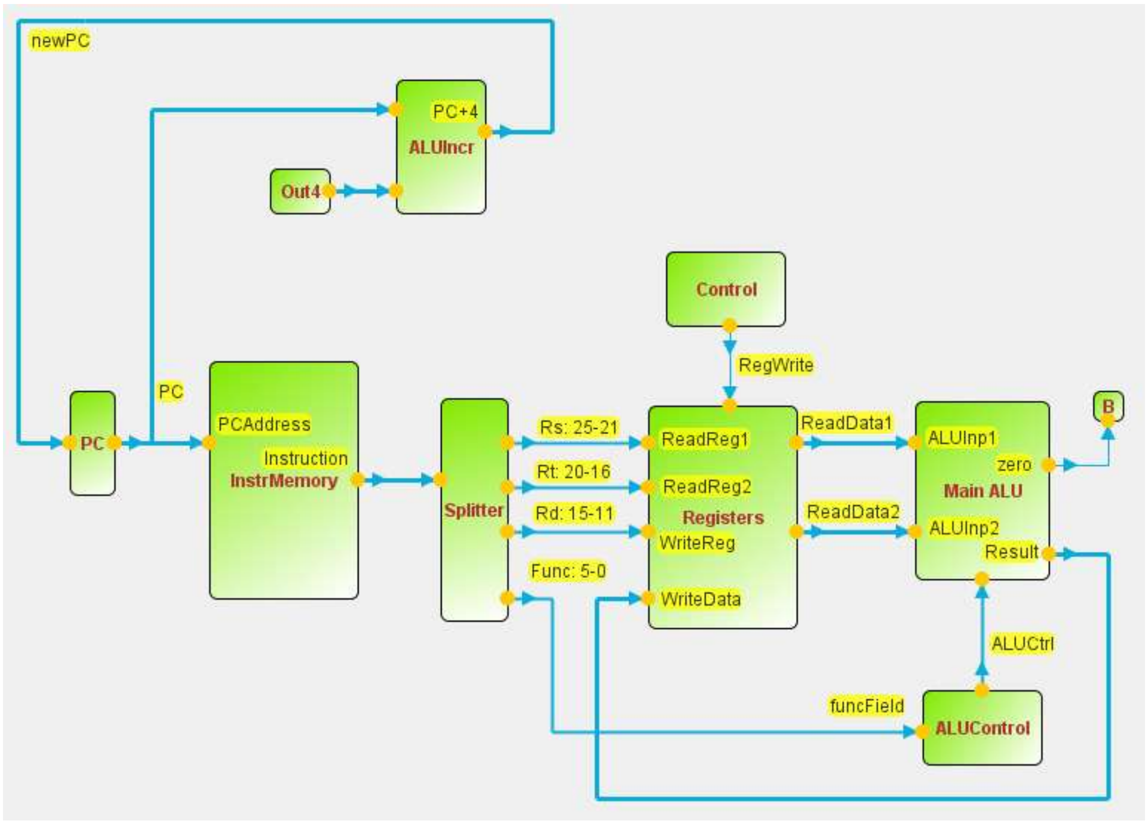


Fig. 2 - First single-cycle datapath example

What you see now is a very simple datapath just supporting the arithmetic R-Format instruction (add, and, or, sub).

1.2 Control Panel:

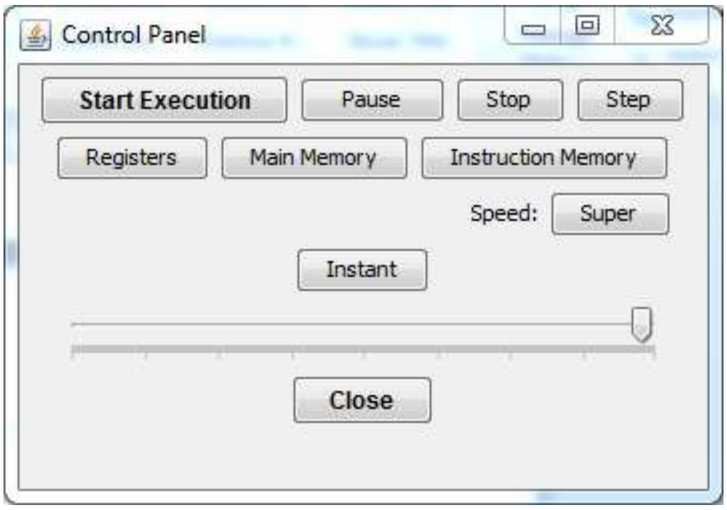
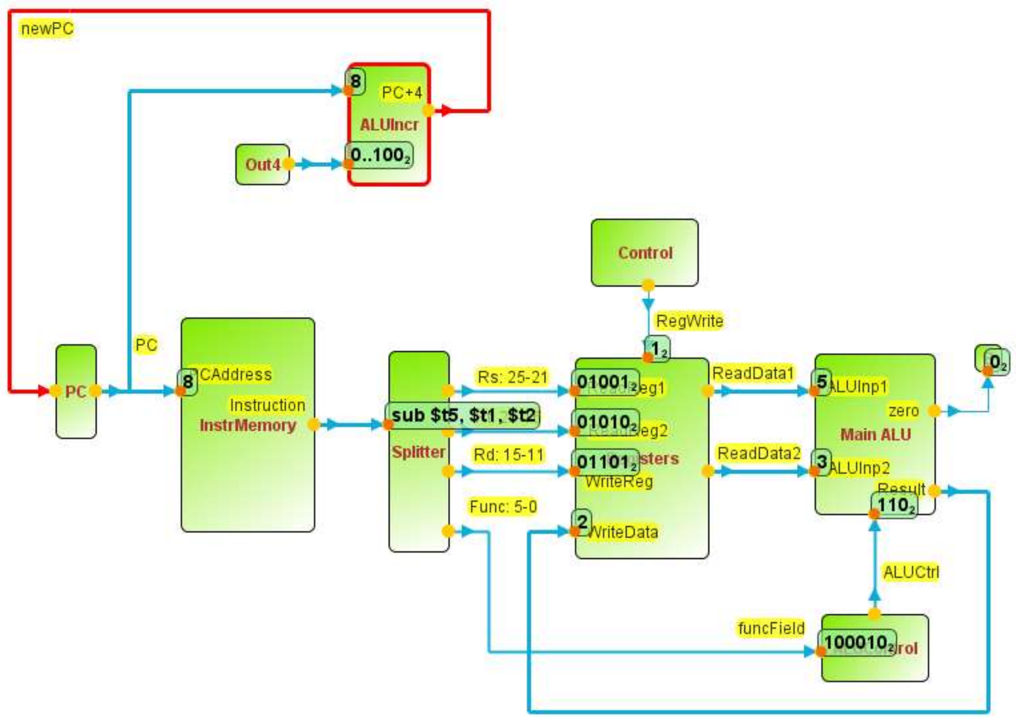
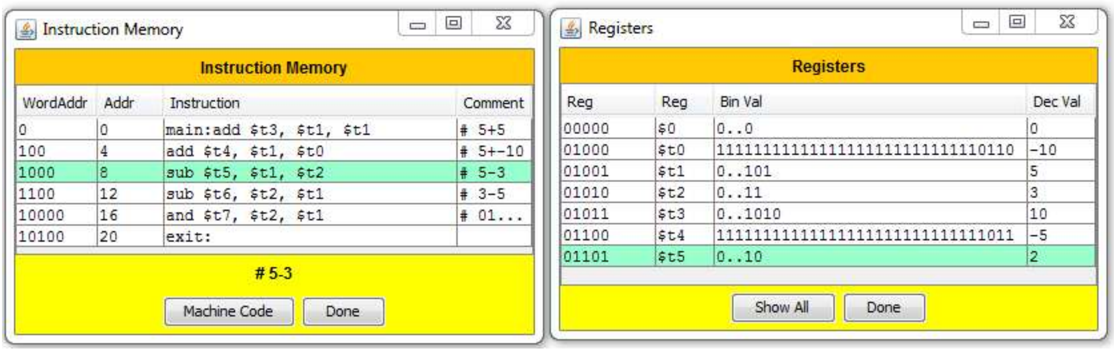


Fig. 3 - Simulation Control Panel

To start the simulation running, you must click "**Start Execution**".





Instruction Memory

WordAddr	Addr	Instruction	Comment
0	0	main:add $t3, $t1, $t1	# 5+5
100	4	add $t4, $t1, $t0	# 5+-10
1000	8	sub $t5, $t1, $t2	# 5-3
1100	12	sub $t6, $t2, $t1	# 3-5
10000	16	and $t7, $t2, $t1	# 01...
10100	20	exit:	

5-3

Registers

Reg	Reg	Bin Val	Dec Val
00000	$0	0..0	0
01000	$t0	11111111111111111111111111110110	-10
01001	$t1	0..101	5
01010	$t2	0..11	3
01011	$t3	0..1010	10
01100	$t4	11111111111111111111111111111011	-5
01101	$t5	0..10	2

Fig. 4 - Simulation Running with datapath animation

You should then see the animation running. You can slow it down by moving the **Speed bar** on the control panel to the left.

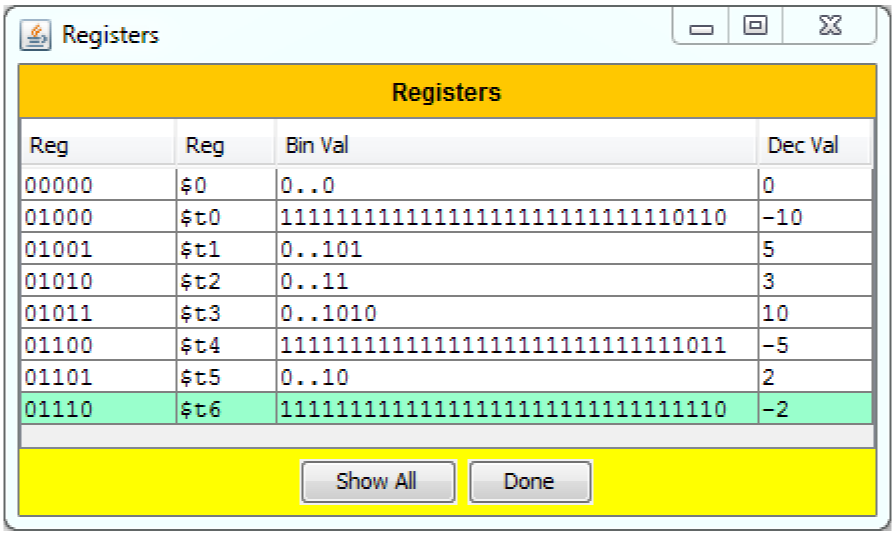
1.3 Other Control Panel Buttons:

Step: Shows one animation step and stops until you click it again, or press resume.
Registers: Shows the Registers frame (which should already be on the screen).
Main Memory: Shows the Main Memory frame.

Instruction Memory:
Shows the Intruction Memory frame (which should already be on the screen).
Super Speed: Only animates at each corner of a bus. So goes very fast.
Instant Speed: Does no animation. If you minimise the simulation window as well, it should almost instantly finish the execution of the assembly program.
Close: Closes the simulation window.

1.4 Registers Window:

Shows all the registers that currently have values stored in them.
The columns are as follows:
Reg - Binary value of the register address,
Reg - String value of the registers,
Bin Val - The value stored in this register in binary, and Dec Val - The value in Decimal.
You can press 'Show All' to see all registers rather than just those with values.
Press 'Done' to hide the window.

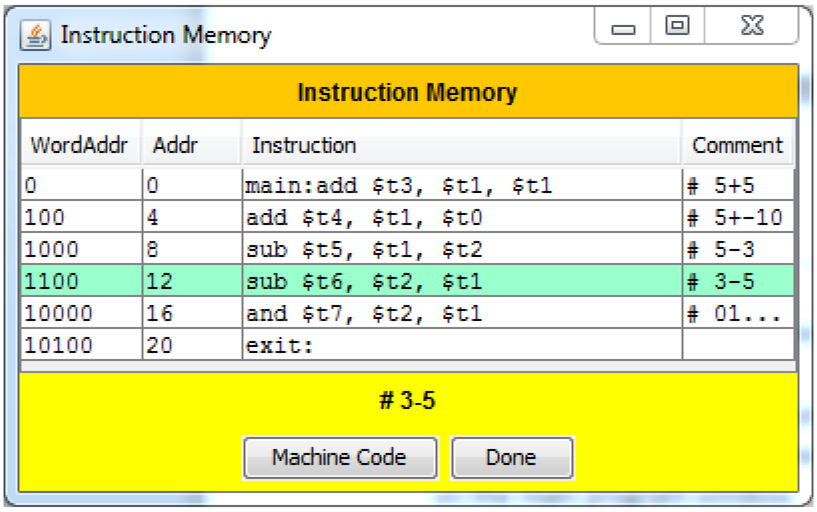


Reg	Reg	Bin Val	Dec Val
00000	$0	0..0	0
01000	$t0	11111111111111111111111111110110	-10
01001	$t1	0..101	5
01010	$t2	0..11	3
01011	$t3	0..1010	10
01100	$t4	11111111111111111111111111111011	-5
01101	$t5	0..10	2
01110	$t6	11111111111111111111111111111110	-2

Fig. 5 - Registers window

1.5 Instruction Memory Window:

This is similar, but here you can choose between seeing the instruction as Assembly or as Machine Code. Each instruction is stored in 4 bytes, so the addresses go up by 4 each time as the MIPS architecture is a byte addressable machine. You can see an instructions comment by clicking on it.



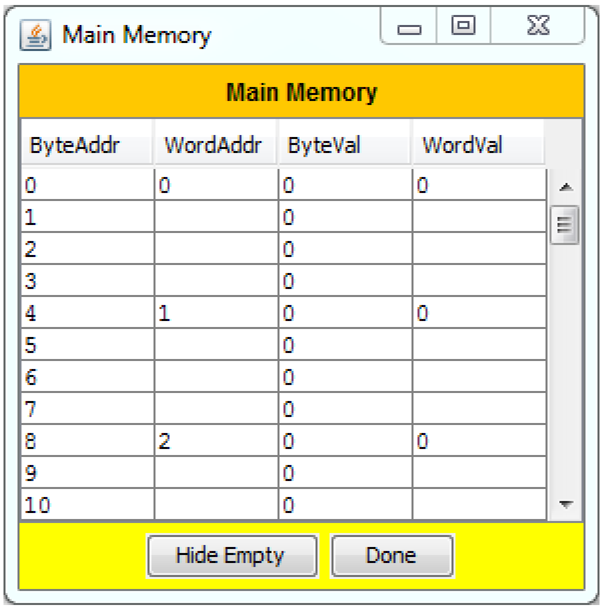
WordAddr	Addr	Instruction	Comment
0	0	main:add $t3, $t1, $t1	# 5+5
100	4	add $t4, $t1, $t0	# 5+-10
1000	8	sub $t5, $t1, $t2	# 5-3
1100	12	sub $t6, $t2, $t1	# 3-5
10000	16	and $t7, $t2, $t1	# 01...
10100	20	exit:	

3-5

Fig. 6 - Instruction Memory window

1.6 Data Memory Window:

This shows each byte's address, as well as each word's address.
This is because the MIPS can choose to read a whole word, or just a byte.



ByteAddr	WordAddr	ByteVal	WordVal
0	0	0	0
1		0	
2		0	
3		0	
4	1	0	0
5		0	
6		0	
7		0	
8	2	0	0
9		0	
10		0	

Fig. 7 - Data Memory window

1.7 Main Frame Menu:

To load a different architecture, you can click "File\Open Sim..." and choose one of the example ".xml" files.
To load an assembly program you can click "File\Open Assembly..." and choose a ".asm" file.
To edit either of these, you must close the sim window and choose the corresponding option on the main program window.

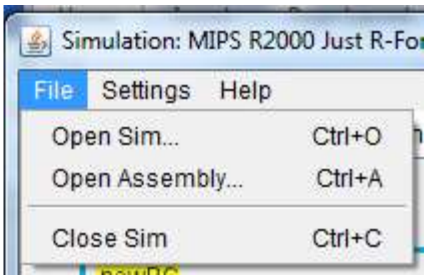


Fig. 8 - Main Frame Menu

1.8 Hints:

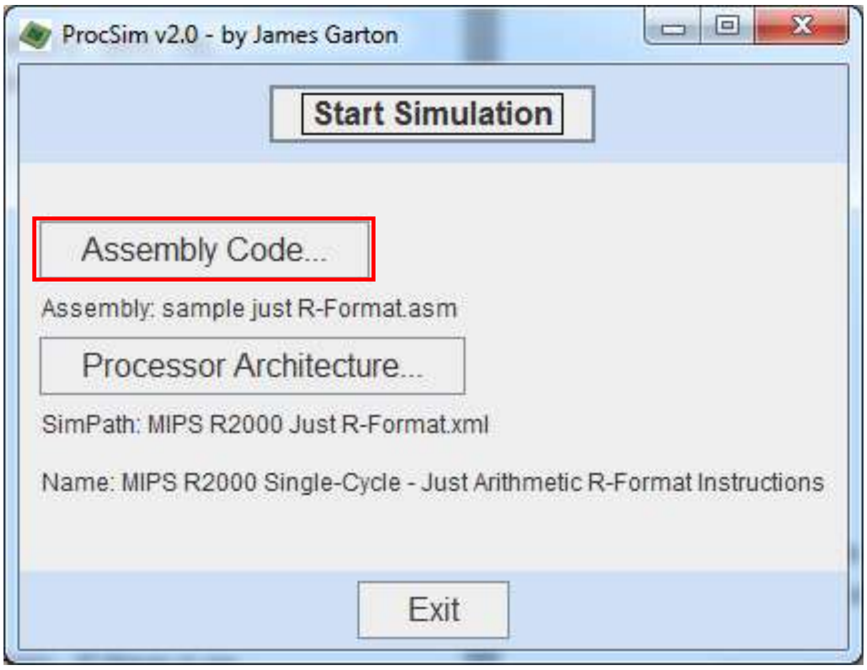
Subscripts: A small subscript 2 on an animating value, means that it is binary. If there is no subscript on the animation then it is decimal.

Description: You can click on a component for a description of what it does. (in the statusbar!)

Animation: If the animation runs too slow or "jerkily", try minimising or closing the other windows. Also turn off Transparency in the settings menu.

Component's execution: A component will only execute if all its input buses are set (they are all red coloured). The only exeption to this is if a bus is optional (which is set in the XML architecture doc).

2. Assembly Code



Assembly: sample just R-Format.asm

SimPath: MIPS R2000 Just R-Format.xml

Name: MIPS R2000 Single-Cycle - Just Arithmetic R-Format Instructions

Fig. 9 - Assembly Code

To edit the assembly code, click "Assembly Code" on the main window. This new screen acts like a text editor.

Supported Assembly:

The supported ISA depends on the currently loaded architecture file.
You can see the supported ISA by clicking "Help\Supported ISA" (these are retrived from the currently loaded XML doc).

Register Directive:

To set registers to default values, you can use an assembler directive:
".register <R1> <value>"
for example ".register $s0 21".

Comments:

Comments can be used. The comment charachter is "#".

Labels:

Labels cannot be placed alone on a line, they have to have an instruction next to them.
The only exception is ":exit" placed at the very end of a program.

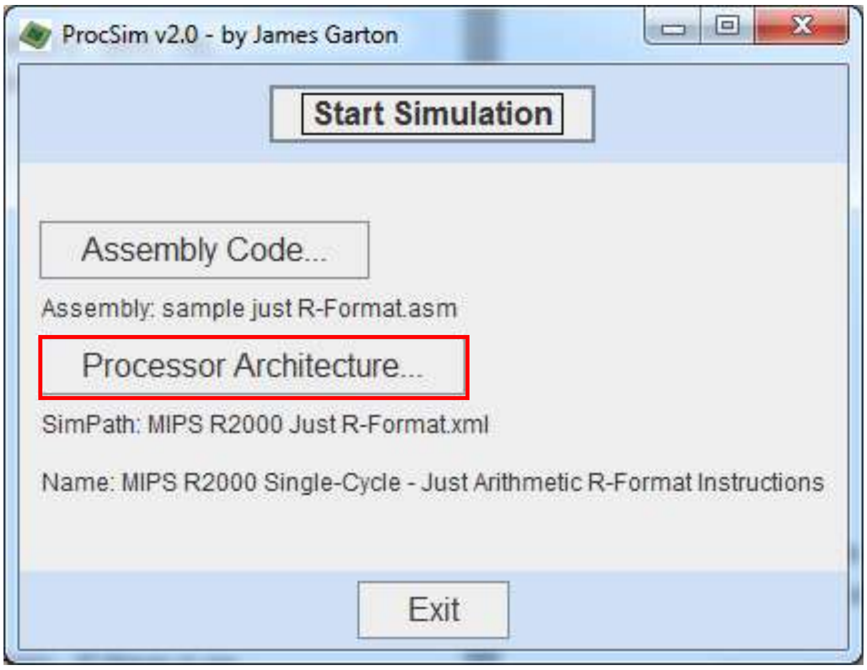
Assembling:

To assemble the code, click Assemble. This will then tell you if your program has been successfully assembled or not.
If not, you can look at the console windows error panel to find where the problem occured.

Machine code can be used instead of assembly.
Only 32bits are supported, and there are other limitations with XML architecture.

3. Architecture Files (XML)

To create different architectures you must create an XML file to detail the components, buses and how they interact. You can use the in program editor to do this. Once you have an XML architecture, you can use the **diagram designer** to draw your architectures components on a canvas (see next paragraph).



Assembly: sample just R-Format.asm

SimPath: MIPS R2000 Just R-Format.xml

Name: MIPS R2000 Single-Cycle - Just Arithmetic R-Format Instructions

Fig. 7 - Data Memory window

First click "Processor Architecture" on the main window. This will show you a window with the currently loaded architectures diagram on it.

Click on "Edit Architecture" to edit the XML. Now you can create/load a new XML doc, or edit the current one. Just look at the examples to work out how to use the XML. All features are used somewhere in the examples.

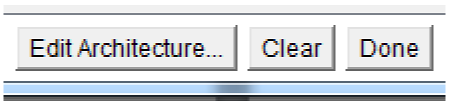
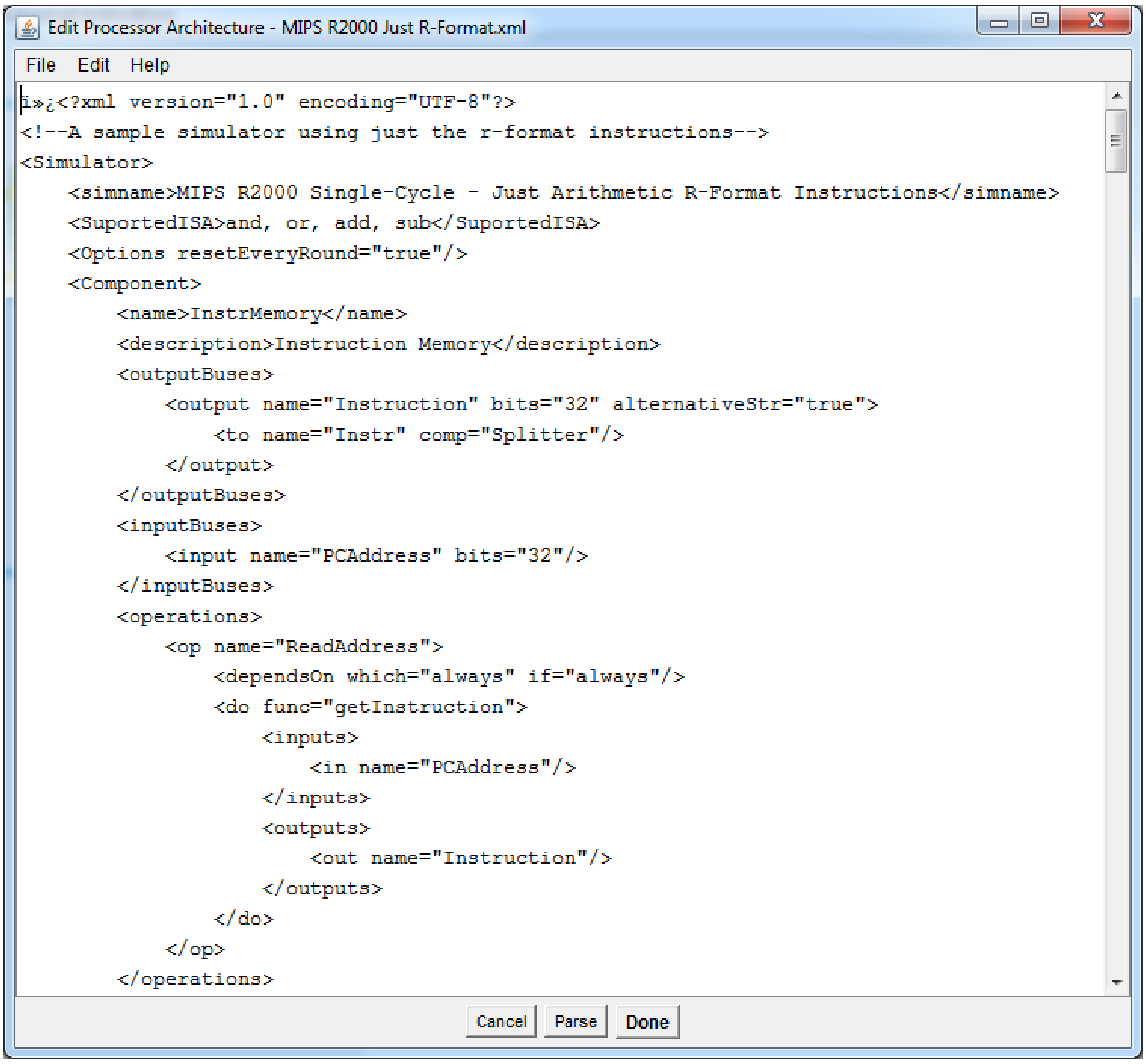


Fig. 8 - 'Edit Architecture...' Button



```
ï»¿<?xml version="1.0" encoding="UTF-8"?>
<!--A sample simulator using just the r-format instructions-->
<Simulator>
    <simname>MIPS R2000 Single-Cycle - Just Arithmetic R-Format Instructions</simname>
    <SuportedISA>and, or, add, sub</SuportedISA>
    <Options resetEveryRound="true"/>
    <Component>
        <name>InstrMemory</name>
        <description>Instruction Memory</description>
        <outputBuses>
            <output name="Instruction" bits="32" alternativeStr="true">
                <to name="Instr" comp="Splitter"/>
            </output>
        </outputBuses>
        <inputBuses>
            <input name="PCAddress" bits="32"/>
        </inputBuses>
        <operations>
            <op name="ReadAddress">
                <dependsOn which="always" if="always"/>
                <do func="getInstruction">
                    <inputs>
                        <in name="PCAddress"/>
                    </inputs>
                    <outputs>
                        <out name="Instruction"/>
                    </outputs>
                </do>
            </op>
        </operations>
```

Fig. 9 - XML Editor window

3.1 Architecture Elements

3.1.1 Do Functions

All supported "<do func="s are as follows:
mux, or, zero, multiplexor, add, and, sub, readmem, writemem, bitout, out, split, readreg, writereg, getInstruction, shiftleft, signextend, slt, join.

Once you have finished creating the architecture, press "Parse". This will tell you if it succedes or fails. If it fails look at the error panel on the console for information about where the problem is.

3.1.2 Hints:

H1: Component musty have at least one op
A component must always be able to do at least one op(eration) if all its input buses are set during an execution.
H2: Buses input/output names are case sensitive
All input and output bus names must match up (case sensitive).

H3: Output Bus Attributes

"<output **name**="Result" **bits**="32" **alternativeStr**="true" **optional**="true">" means that this output buses *name* is *Result* and it has *32* bits.
The other attributes are optional:
alternativeStr means that the bus will show instead of a binary vale, a decimal value (or assembly in the case of an instruction).
optional means that this bus does not have to be set for the connected component to do its op(erations)s.

H4: Minimum Component XML Description

The least a component must have is the following:

```
<Component>
        <name> MyNameA</name>
        <description> MyNameA Description</description>
        <inputBuses>
                <input name="Z" bits="1"/>
        </inputBuses>
</Component>
```

(The input Z must be connected up to an outputbus from a different component - which may only have an output).

H5: Special Component

<specialComponent comp="StartComp"/> means that this component will be the first to start executing. It will set all its output buses to 0 when starting execution. Mostly used for PC.

4. Architecture Diagram

Once you have created an XML architecture document you can create the diagram to go with it. To edit this, click on Processor Architecture on the main screen. You can now see the currently loaded architecture. (Use the File/Load menu if you need to load an XML doc you did before).

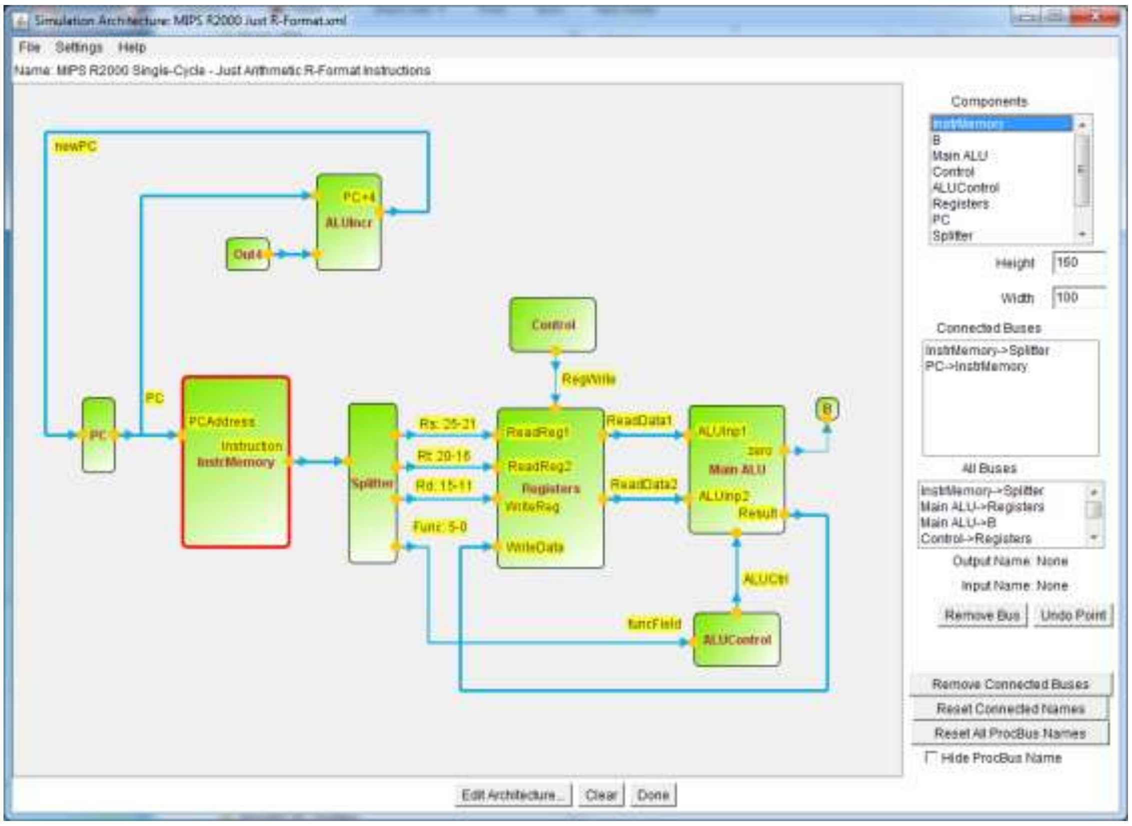


Fig. 10 - The Datapath Diagram Designer window

Add Components

You can now **add the components** to the diagram. Just click on one of the items in the list in the top right. Then click anywhere on the canvas. A square box will be placed dow n. You can drag it around the screen. Change its height and width in the boxes on the right.

Draw Buses

Once you have placed some components down, you can **start drawing the buses**. To do this click on the component you wish to add buses to, then click on a bus in the middle list on the right ("Connected buses"). You will now see the two component that this bus joins together change colour. The red component is the output bus, and blue is input. Click on the canvas anywhere close to the output component to place the bus on it. Now you can keep clicking to place more lines down (all joined up). Click on the input component to finish editing that bus.

Arrange Buses Names

Now you will see some **bus names**. You can move them too. You can also hide any bus names you feel are irelevant by clicking "Hide Bus Names" on the right bottom.

When you are happy with the diagram, you can press **Done**.

For more help look through the extensive set of examples.

James M Garton 18/09/2005
james@jamesgart.com
http://jamesgart.com/procsim