ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Special Issue 3, April 2017

Simulator for MIPS Processor

P.Kopperundevi[1], S.Muthukumar[2]

PG Scholar, Department of ECE, SVCE, Sriperumbudur, India , [1].

Professor, Department of ECE, SVCE, Sriperumbudur, India, [2].

ABSTRACT: A MIPS is a version of RISC processor. A new tool for MIPS-32 processor simulation has been designed and developed. MIPS simulator is based on java language. This tool is mainly intended for educational use to teach computer architecture and test the working of MIPS assembly language programs. The Programming model incorporating all registers is included in the design. A menu driven approach is used. The tool is designed in such a way that it is easy to use. GUI is rendered in the front end. It supports syntax highlighting process which makes it easier to deal with multiple commands, variable and comments. Although the MIPS IDE is clearly designed for programmers and MIPS developers, it includes features such as command description, spread sheet view of registers, which can help the less experienced one. The tool that is developed can be used to read and execute assembly language programs written for MIPS processor. The registers and memory values can be easily edited and values are displayed either in hexadecimal, decimal or binary. Assembly language programs can be easily tested and evaluated. Integer registers, floating point registers co-processor are included in the design. Therefore as a tool for students it has a large potential value

KEYWORDS: Assembly language, MIPS

I.INTRODUCTION

The MIPS is a RISC architecture and corresponding assembly language use a limited number of instruction formats. Typical student programs may use register-to-register, load/store, branch, jump, system call, and floating-point instructions. Thirty-two general-purpose registers are available for integer operations (some have dedicated uses), as are thirty-two single-precision floating point registers. MIPS-32 is a clean design with simple instructions. Since computer science and computer engineering departments may not have adequate access to MIPS equipment to support laboratory activities, software-based MIPS simulators may be used. MIPS simulator is designed as an alternative to SPIM specifically for the needs of typical undergraduate students and their instructors. It should be useful in courses such as computer organization and architecture, assembly language programming, and compiler writing.

A.MIPS Simulator Features:

MIPS simulator is an Integrated Development Environment (IDE) controlled by a modern GUI whose features include

- Thirty-two registers visible at the same time, selectable via tabbed interfaces,
- “Spreadsheet” modification of values in registers and memory,
- Selection of data value display in decimal or hexadecimal,
- Resizable windows,
- “Surfing” through memory using buttons to change display to next/previous, stack location, global partition, and the start of the memory segment,

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Special Issue 3, April 2017

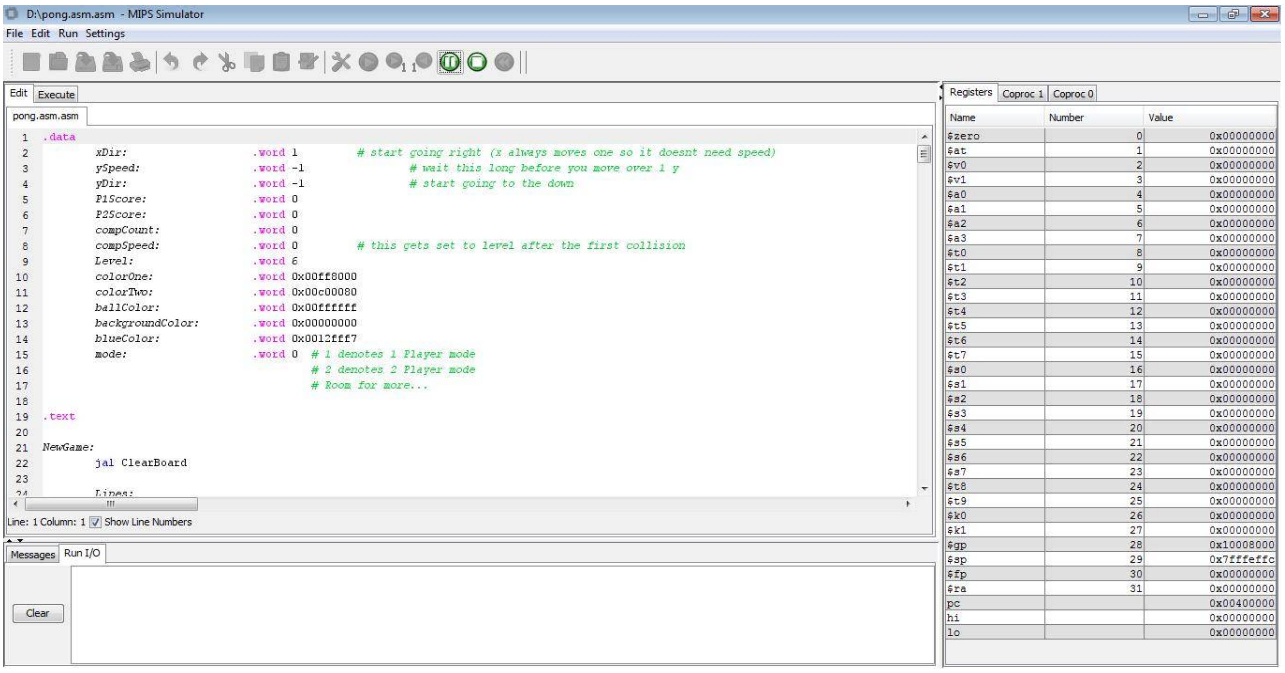
- Toolbar icons for every menu item
- An integrated editor and assembler as part of its IDE.

The MIPS simulator implements the educationally important portions of the MIPS instruction set utilized by ComputerOrganization and Design Third Edition (COD3)[7].

II.RELATED WORK

A. Simulator – Based on Vhdl Language:

S. P. Ritpurkar et al (2014), Synthesis and Simulation of a 32Bit MIPS RISC Processor using VHDL[8]. In this paper, they have analyzed Instructionfetch module, Decoder module, Execution module which includes32Bit Floating point ALU, Flag register of 32Bit, MIPS Instruction Set, and 32Bit general purpose registers and designtheory based on 32Bit MIPS RISC Processor. Menu deriven approach is used. Floating point and integer registers are used .. In terms of performance data, the total clock cycles are provided. Furthermore,pipeline concept is used which involves Instruction Fetch,Instruction Decode, Execution, Memory and Write Back modulesof MIPS RISC processor based on 32Bit MIPS Instruction set ina single clock cycle. All the modules in the design are coded inVHDL, as it is a very useful language with its concept ofconcurrency to cope successfully with the parallelism of digitalhardware. Finally, Synthesis and Simulation of the design is donein XILINX 13.1i ISE simulator. performance is verified using cadence tool both analog and digital view and result are verified.



```
.data
        xDir:                   .word 1         # start going right (x always moves one so it doesnt need speed)
        ySpeed:                 .word -1                # wait this long before you move over 1 y
        yDir:                   .word -1                # start going to the down
        P1Score:                .word 0
        P2Score:                .word 0
        compCount:              .word 0
        compSpeed:              .word 0         # this gets set to level after the first collision
        Level:                  .word 6
        colorOne:               .word 0x00ff8000
        colorTwo:               .word 0x00c00080
        ballColor:              .word 0x00ffffff
        backgroundColor:        .word 0x00000000
        blueColor:              .word 0x0012fff7
        mode:                   .word 0  # 1 denotes 1 Player mode
                                         # 2 denotes 2 Player mode
                                         # Room for more...

.text

NewGame:
        jal ClearBoard

        Lines:
```

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

Figure 1. MIPS Simulator Editor Window is Active ("Edit" Tab is Foremost)

B. Simulator – Based on Instruction Set Architecture:

WinMIPS 64, EduMIPS 64, and Simple MIPS [9] Pipeline are simulators for pipeline processors. They focus on modeling functional aspects of pipeline stages instead of RT level logic components inside processors. EduMIPS64 is actually a re-design and re-implementation of WinMIPS 64 in Java. MiniMIPS has the similar goal as this work. However, it models control units at a higher level instead of treating them as the composition of RT level components,

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Special Issue 3, April 2017

such as shifters, Arithmetic Logic Unit (ALU) controls, and multiplexors. These lower level logic components are important for understanding processor implementations. Also, their attributes, such as delay, are needed to model processor performances. From the limited resources that obtained from the authors, it seems that MiniMIPS does not provide animation, only provides cycle count as performance data, and is implemented in C and requires Unix machines.

WebMIPS[2] only models a pipeline processor. It models all the components inside the processor, and users can view each component's input and output data at a certain time by clicking on the component. However, the simulator does not show how the signals are sent and received among components during instruction execution. In terms of performance data, the total clock cycles are provided. Thus WEBMIPS has limited number of users. It can manage applications based on the online not on offline view. Its simple to use and being graded. Memory reference visualize tool is used. It does not have pipeline concept

ProcessorSim can be configured to model several data path configurations for the MIPS-32 single-cycle processor implementation and provides an animation that shows how instructions are executed inside processors. It provides good visualization but has some shortcomings. In ProcessorSim, only one component can send out messages at a time. However, components inside a real processor always work concurrently. ProcessorSim only shows effective execution paths for an instruction execution. That makes it easier for students to understand the processor implementations but hides some important details. ProcessorSim does not model component delays and thus can only support limited performance data. ProcessorSim is not based on any modeling and simulation theory and therefore lacks a rigorous basis for defining the structures and behaviors of the MIPS components and their compositions. Thus, extending ProcessorSim to support other processor designs (e.g. MIPS 64) is difficult.

III.**MIPS SIMULATOR OPERATION**

The MIPS simulator operates under either GUI or command-line modes of operation. MIPS simulator is used in use the GUI mode in either “go” or “single step” execution for assembly code creation and debugging. Instructors have the option of running the simulator from an OS shell or a batch command file, to facilitate execution of several test cases of all student’s programs in sequence for grading. Command-line arguments are used to request the output of particular registers or memory locations to verify program results. The MIPS simulator is written in Java 1.4.2, using standard techniques of human-computer interaction via its Swing and AWT packages. Standard icons have been obtained from the Java look and feel Graphics Repository [10].MIPS Simulator is used to compose an assembly language program using the editor, assemble it, then execute the assembled program all at once or step-by-step using the facilities of the execute pane. These operations are illustrated and described here. The MIPS simulator editor is an ASCII-oriented text editor that operates much like Window's Notepad. Figure 1 shows the active editing panel. The first two groups of toolbar icons are used with the editor. The first group corresponds to the File menu and includes file options such as New, Open, and Save. The second group corresponds to the Edit menu and includes operations such as Cut, Copy and Paste. Menu items and their corresponding toolbar buttons are enabled and disabled as appropriate. To assemble the program, the user selects Assemble from the Run menu or clicks the wrench toolbar icon. A successful assemblycauses the execute pane to come forward as shown in Figure 2.An unsuccessful assembly displays appropriate messages and linenumbers in the console window at the bottom of the screen.

A.Text Segment:

The Execute pane contains several windows. The Text Segment window is front and center. It displays both the source and binary code of the assembly program, including the expansion of pseudoinstructions (the lw and jal instructions in Figure 2). A breakpoint can be set at any instruction using the check box in the leftmost column. When stepping through program execution manually or at reduced run speeds, the next instruction to be executed is highlighted.

B.Data Segment:

The Data Segment display illustrated at the bottom of Figure 2 shows the program's data storage area in a

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Special Issue 3, April 2017

scrollable window. Its lower border contains icons to control display of memory contents at special locations such as the stack or heap, and check boxes to display memory addresses and values in either decimal or hexadecimal format. Symbol table information is displayed in the Labels window. This is relatively less important and the window may be closed to allow more space for the Text Segment display. Registers are permanently displayed to the right of the Execute pane in a vertically oriented window. This can be seen in the right side of Figure 1. As with memory, values are editable and display format is selectable. There are separate tabs for the general purpose registers, the floating point registers of Coprocessor 1 and the exception registers of Coprocessor 0. Another permanent display is the console window on the lower portion of the screen. It includes two tabs, one for MIPS messages such as assembly errors and another for runtime input and output generated by MIPS system calls. Each tab is activated when text is written to it. When starts programming it assemble the file automatically via settings default and assembly errors can be avoided or considered as warnings or else it can be avoided, thus based on settings modification takes place.

C.Simulator Categories:

A number of MIPS simulators have been developed over the years. Most can be classified by a small number of categories: those designed for research use (e.g. MIPSI), those that focus on certain MIPS architectural features such as pipelines (e.g. WebMIPS [2], SmallMIPS, RTLSim), those that depend on SPIM (e.g. MIPSASM, TinyMIPS), and general purpose simulators. Examples of the latter include MipsIt and SPIM MIPS simulators include features for visualizing and/or animatingMIPS components. MIPS simulator and SPIM do not.

D.MIPS Simulator Comparision With Spim:

A comparison of some education oriented characteristics of SPIM to MIPS simulator follows.

The SPIM user interface has one window split into five Scrollable but non-resizable panes. Using PCSPIM on a 19” monitor, at most nine lines of source code are visible at a time. MIPS simulator uses resizable windows and tabbed panes to more easily focus on memory, register or program contents.

Several steps are required to modify register or memory values in SPIM: calling up a pop-up window, typing the register or address, and specifying the new value. This is time-consuming and error-prone. MIPS simulator features allow on-the-spot modification.

Similarly, SPIM’s breakpoints are set by calling up a pop-up window and typing the breakpoint location. MIPS simulator features a check box beside each line of code to set and remove breakpoints immediately.

Registers are permanently displayed to the right of the Execute pane in a vertically oriented window. This can be seen in the right side of Figure 1. As with memory, values are editable and display format is selectable. There are separate tabs for the general purpose registers, the floating point registers of Coprocessor1 and the exception registers of Coprocessor0.Another permanent display is the console window on the lower portion of the screen. It includes two tabs, one for MIPS simulator messages such as assembly errors and another for runtime input and output generated by MIPS system calls. Each tab is activated when text is written to it. The MIPS simulator text editor currently provides Notepad-like functionality. Some contextual help is provided by tool tips that appear when the mouse is hovered over the always-present Register window. The two aspects introduced in MIPS simulator implementation represent its larger contribution: external instruction set specification, and tool plug-in capability. Both are partially achieved at this time. The simplicity and regularity of the MIPS instructions permit the separation of the specification of MIPS simulator instructions from other source codes

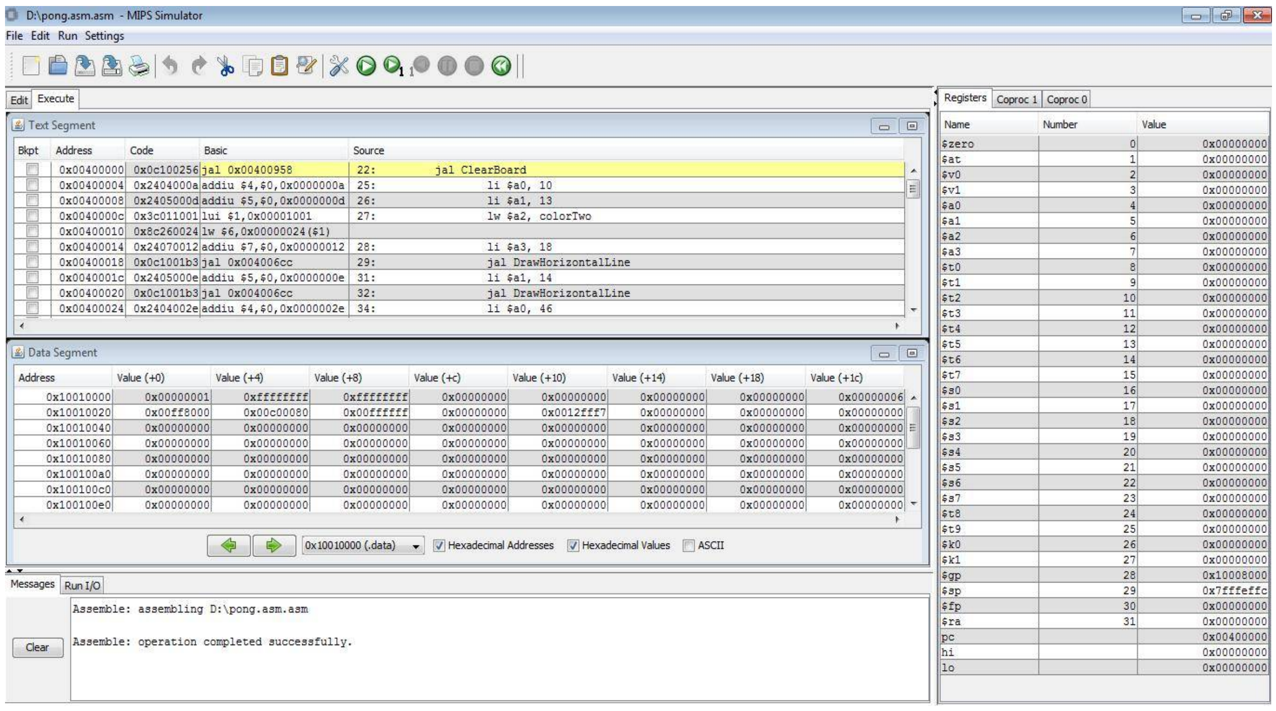
ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Special Issue 3, April 2017



Bkpt	Address	Code	Basic	Source
☐	0x00400000	0x0c100256	jal 0x00400958	22: jal ClearBoard
☐	0x00400004	0x2404000a	addiu $4,$0,0x0000000a	25: li $a0, 10
☐	0x00400008	0x2405000d	addiu $5,$0,0x0000000d	26: li $a1, 13
☐	0x0040000c	0x3c011001	lui $1,0x00001001	27: lw $a2, colorTwo
☐	0x00400010	0x8c260024	lw $6,0x00000024($1)	
☐	0x00400014	0x24070012	addiu $7,$0,0x00000012	28: li $a3, 18
☐	0x00400018	0x0c1001b3	jal 0x004006cc	29: jal DrawHorizontalLine
☐	0x0040001c	0x2405000e	addiu $5,$0,0x0000000e	31: li $a1, 14
☐	0x00400020	0x0c1001b3	jal 0x004006cc	32: jal DrawHorizontalLine
☐	0x00400024	0x2404002e	addiu $4,$0,0x0000002e	34: li $a0, 46

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000001	0xffffffff	0xffffffff	0x00000000	0x00000000	0x00000000	0x00000000	0x00000006
0x10010020	0x00ff8000	0x00c00080	0x00ffffff	0x00000000	0x0012fff7	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

0x10010000 (.data) [✓] Hexadecimal Addresses [✓] Hexadecimal Values ☐ ASCII

```
Assemble: assembling D:\pong.asm.asm

Assemble: operation completed successfully.
```

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

Figure 2. MIPS Simulator Execution Window is Active ("Execute" Tab is Foremost and the Execution Toolbar Icons are Active)

The two aspects introduced in MIPS simulator implementation represent its larger contribution: external instruction set specification, and tool plug-in capability. Both are partially achieved at this time. The simplicity and regularity of the MIPS instructions permit the separation of the specification of MIPS simulator instructions from other source codes.

The specification for each instruction consists of:
· an example usage of the instruction
· the instruction format
·a template of the generated 32 bit machine instruction with operand positions
indicated · a Java method to simulate the execution of the instruction

All except the last are strings that may be placed in a textual configuration file for loading when MIPS simulator is launched. Similarly,a separate text file is used to specify MIPS "pseudo-instructions" (a.k.a. macro instructions). Pseudo-instructions are expanded into one or more native MIPS instructions by the assembler. For each pseudo-instruction, the text file contains a specification consisting of an example usage followed by a tab-separated list of native instructions into which it will be translated with appropriate operand substitution.MIPS memory locations and reacts appropriately in response to data changes in the memory-mapped IO locations defined for this tool.

IV.MIPS SIMULATOR TOOLS

The tool plug-in capability permits the definition of customized bots, animations, or any number of other useful tools to be controlled by a MIPS program during MIPS simulation**.** The tool plug-in capability permits the definition of customized bots, animations, or any number of other useful tools to be controlled by a MIPS program during MIPS simulation. A tool "observes"

MIPS memory locations and reacts appropriately in response to data changes in the memory-mapped IO locations defined for this tool. The source code of a tool is separate from the source code of MIPS simulator. Using a dynamic

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Special Issue 3, April 2017

class-loading technique from game programming [1], any externally-compiled class which implements a certain Java interface and resides in the tools folder will be detected and loaded at MIPS simulator launch and added to its Tools menu (see Figure 1). User selection of that Tools menu item will invoke a particular interface method, which will typically establish itself as an observerof MIPS memory locations. A MIPS program will read and write memory locations and the tool will respond accordingly.

V.RESULTS AND CONCLUSION

Traditionally, students use a text editor to generate lines of code for use in the SPIM simulator located on the ECE machines or using other window based simulators. The problem with this approach is there is no feedback given to the student when writing the code. When loading the code into the simulator, feedback on any errors is difficult to discern or understand This can create a problem for students who are new to the language, and frustration when trying to determine the cause of an error. The use of the MIPS simulator alleviates this problem by use of a power interactive development environment (IDE) that can help students understand the code they are writing. MIPS simulator implements 98 MIPS instructions, 32 native instructions, 36 pseudo-instructions, and the 17 system calls. The Edit and Execute tab which is designed for MIPS processor is shown in figure 1 and Figure 2 respectively.

VI.FUTURE WORK

Other plans include implementing the remaining instruction set, improving debugging support through such features as highlighting of memory/register contents modified in step-by-step execution, the ability to undo execution steps and interfacing.

REFERENCES

1. Brackeen, David, Barker, Bret, and Vanhelswue, Laurence, "Developing Games in Java". New Riders Publishing, 2015.
2. Branovic, I., Giorgi, R. and Martinelli, E., WebMIPS : A New Web-Based MIPS Simulation Environment for Computer Architecture Education, Workshop on ComputerArchitecture Education, 31st International Symposium on Computer Architecture, Munich, Germany, 2016. for Computer Architecture Education, Workshop on Computer Architecture Education, 29th International Symposium on Computer Architecture, Anchorage AK, 2014.
3. Downcast Systems, MIPS 2.0, http://www.downcastsystems.com/MIPS ter/, retrieved 21 November 2005

J.Garton. ProcessorSim - A visual MIPS R2000 processor simulator. http://jamesgart.com/procsim/, 2005
Larus, J., SPIM: An MIPS32 simulator, http://www.cs.wisc.edu/~larus/spim.html, retrieved 21 November 2005.

6. N. Mohit Topiwala, N. Saraswathi : Implementation of a 32-bit MIPS-Based RISC Processor using CadenceIEEE International Conference on Advanced Communication Control and Computing Technologies (ICACCCT) ISBN No. 978-1-4799-3914-5/14/$31.00 ©2014
7. Patterson, D., and Hennessy, J., Computer Organization and Design: The Hardware/Software Interface, 3rd edition, SanFrancisco, CA: Morgan Kaufmann, 2004
8. S. P. ritpurkar prof., M. N. thakare. prof. G. D. korde :Synthesis and simulation of a 32bit MIPS RISC processor using VHDL on International conference on advances in engineering & technology research (icaetr - 2014), august 01-02, 2014.

Sun Microsystems, Java look and feel GraphicsRepository, http://java.sun.com/developer/techDocs/hi/repository/, retrieved 21 November 2005.

Vollmar, K., and Sanssderson, P., A MIPS Assembly Language simulator Designed For Education. The Journal ofComputing Sciences in Colleges, Vol. 21, No. 1, 2005.

11. Wolffe,G.,Yurcik, W.,Obsborn,h. and Holiday, M.,teaching computer architechture/organization with limited resources,AC SIGCSE Bulliten 34,(1),176-180,2016
12. Mrs. Rupali S. Balpande, Mrs.Rashmi S. Keote, Design of FPGA based Instruction Fetch & Decode Module of 32-bit RISC (MIPS) Processor, 2011 International Conference on Communication Systems and Network Technologies, 978-0-7695-4437-3/11, 2011 IEEE.
13. Mamun Bin Ibne Reaz, MEEE, Md. Shabiul Islam, MEEE, Mohd. S. Sulaiman, MEEE, A Single Clock Cycle MIPS RISC Processor Design using VHDL, ICSE2002 Proc. 2002, Penang, Malaysia, 0-7803-7578- S/02/S, 2002 IEEE.
14. Kui YI, Yue-Hua DING, 32-bit RISC CPU Based on MIPS Instruction Fetch Module Design, 2009 International Joint Conference on Artificial Intelligence, 978-0-7695-3615-6/09, 2009 IEEE.
15. Rohit Sharma, Vivek Kumar Sehgal, Nitin Nitin1, Pranav Bhasker, Ishita Verma, Design and Implementation of a 64-bit RISC Processor using VHDL, UKSim 2009: 11th International Conference on Computer Modelling and Simulation, 978-0-7695-3593-7/09, 2009 IEEE.
16. Pravin S. Mane, Indra Gupta, M. K. Vasantha, Implementation of RISC Processor on FPGA, 1-4244-0726-5/06, 2006 IEEE.
17. Ardsher Ahmed, Pat Conway, Bill Hughes, and Fred Weber. AMD Opteron Shared Memory MP Systems. In *Proceedings of the 14th HotChips*

Symposium, August 2002.

18. Homayoon Akhiani, Damien Doligez, Paul Harter, Leslie Lamport, Joshua Scheid, Mark Tuttle, and Yuan Yu. Cache Coherence Verification with TLA+. In *FM'99—Formal Methods, Volume II*, volume 1709 of *Lecture Notes in Computer Science*, page 1871. Springer Verlag, 1999.
19. Alaa R. Alameldeen, Milo M. K. Martin, Carl J. Mauer, Kevin E. Moore, Min Xu, Daniel J. Sorin, Mark D. Hill, and David A. Wood. Simulating a $2M Commercial Server on a $2K PC. *IEEE Computer*, 36(2):50–57, February 2003.
20. [4] Todd Austin, Eric Larson, and Dan Ernst. SimpleScalar: An Infrastructure for Computer System Modeling. *IEEE Computer*, 35(2):59–67, February 2002.
21. Nohl A, Braun G, Schliebusch O, Leupers R, Meyr H, HoffmannA (2002) A universal technique for fast and flexible instruction-setarchitecture simulation. In: Proceedings of the design automationconference (DAC 2002), pp 22–27.
22. Reshadi M, Mishra P, Dutt N (2003) Instruction set compiled simulation:A technique for fast and flexible instruction set simulation.In: Proceedings of the design automation conference (DAC 2003),pp 758–763Lee J, Kim J, Jang C, Kim S, Egger B, Kim K, Han SY (2008) FaCSim:Afast and cycle-accurate architecture simulator for embeddedsystems. In: Proceedings of the conference on languages, compilers,and tools for embedded systems (LCTES'08), pp 89–99.
23. Bartholomeu M, Azevedo R, Rigo S, Aruajo G (2004) Optimizationsfor compiled simulation using instruction type information.In; Proceedings of the symposium on computer architecture andhigh, performance computing (SBAC-PAD'04), pp 74–81.
24. D'Errico J, Qin W (2006) Constructing portable compiledinstruction-set simulators: an ADL-driven approach. In: Proceedingsof the conference on design, automation and test in Europe(DATE 2006), pp 112–117.
25. Mong WS, Zhu J (2004) DynamoSim: a trace-based dynamicallycompiled instruction set simulator. In: Proceedings of the internationalconference on computer aided design (ICCAD 2004),pp 131–136.