International Journal of Emerging Technology in Computer Science & Electronics (IJETCSE)
ISSN: 0976-1353 Volume 24 Issue 9 – JUNE 2017.

AN APPLICATION ORIENTED INTERFACING OF KEYBOARD WITH MIPS SIMULATOR

P.Kopperundevi
ME- *Applied Electronics*
Department of ECE
Sri Venkateswara College of Engineering,India

S.Muthukumar
Professor
Department of ECE
Sri Venkateswara College of Engineering,India

***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. . It supports syntax highlighting process which makes it easier to deal with multiple commands, variables 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 integrated art of interfacing of MIPS processor with its simulator is discussed in this paper. The tools are designed in such a way that it is easy to use. Bitmap display and keyboard and MMIO simulator are the tools used for interfacing. As an application of keyboard interfacing master snake game is designed and simulated based on MIPS assembly language using MIPS simulator.**

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,
- 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 Computer Organization 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. In terms of performance data, the total clock cycles are provided. Instruction set ina single clock cycle. All the modules in the design are coded inVHDL,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.

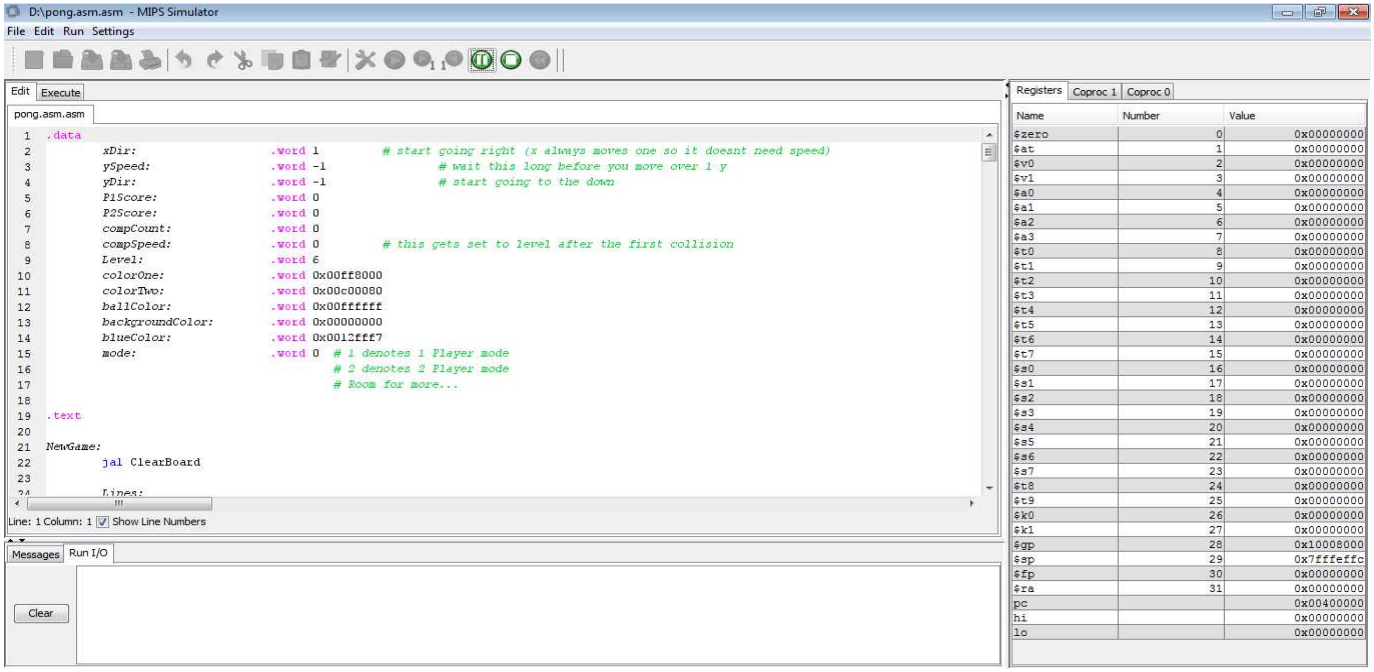

Figure 1: MIPS Simulator Editor Window is Active

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, 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 are 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 TOOLS

Keyboard and MMIO simulator and bitmap display used to interface keyboard .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 class-loading technique from game programming, 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 observer of MIPS memory locations. A MIPS program will read and write memory locations and the tool will respond accordingly.

A. Keyboard and MMIO simulator

The keyboard and MMIO simulator is used to simulate Memory-Mapped I/O (MMIO) for a keyboard input device and character display output device. It may be run either from MARS' Tools menu or as a stand-alone application. While the tool is connected to MIPS, each keystroke in the text area causes the corresponding ASCII code to be placed in the Receiver Data register (low-order byte of memory word 0xffff0004), and the Ready bit to be set to 1 in the Receiver Control register (low-order bit of 0xffff0000). The Ready bit is automatically reset to 0 when the MIPS program reads the Receiver Data using an 'lw' instruction.A program may write to the display area by detecting the Ready bit set (1) in the Transmitter Control register (low-order bit of memory word 0xffff0008), then storing the ASCII code of the character to be displayed in the Transmitter Data register (low-order byte of 0xffff000c) using a 'sw' instruction. This triggers the simulated display to clear the Ready bit to 0, delay awhile to simulate processing the data, then set the Ready bit back to 1. The delay is based on a count of executed MIPS instructions.

In a polled approach to I/O, a MIPS program idles in a loop, testing the device's Ready bit on each iteration until it is set to 1 before proceeding. This tool also supports an interrupt-driven approach which requires the program to provide an interrupt handler but allows it to perform useful processing instead of idly looping. When the device is ready, it signals an interrupt and the MIPS simulator will transfer control to the interrupt handler.Interrupt-driven I/O is enabled when the MIPS program sets the Interrupt-Enable bit in the device's control register.

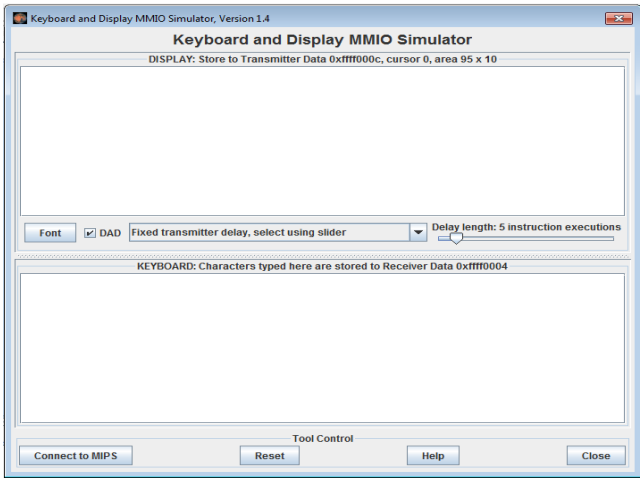

Figure 2: Keyboard and MMIO Simulator

Upon setting the Receiver Controller's Ready bit to 1, its Interrupt-Enable bit (bit position 1) is tested. If 1, then an External Interrupt will be generated. Before executing the next MIPS instruction, the runtime simulator will detect the interrupt, place the interrupt code (0) into bits 2-6 of Coprocessor 0's Cause register ($13), set bit 8 to 1 to identify the source as keyboard, place the program counter value (address of the NEXT instruction to be executed) into its EPC register ($14), and check to see if an interrupt/trap handler is present (looks for instruction code at address 0x80000180). If so, the program counter is set to that address. If not, program execution is terminated with a message to the Run I/O tab. The Interrupt-Enable bit is 0 by default and has to be set by the MIPS program if interrupt-driven input is desired. Interrupt-driven input permits the program to perform useful tasks instead of idling in a loop polling the Receiver Ready bit! Very event-oriented. The Ready bit is supposed to be read-only but in MIPS it is not.

B. Bitmap Display

Use this program to simulate a basic bitmap display where each memory word ina specified address space corresponds to one display pixel in a row-major order starting at the upper corner of the display. The tool may be run either from the MIPS simulator tools menu or as a standalone application. Each rectangular unit on the display represents one memory word in a contiguous address space starting with the specified base address .the value stored in that word will be interpreted as a 24-bit RGB color value with a red component in a bits 16-23,thegreen component in bits 8-15,and blue component in bits 0-7.each time a memory word within the display address space is written by the MIPS program, its position in the display will be rendered in the color that its value represents

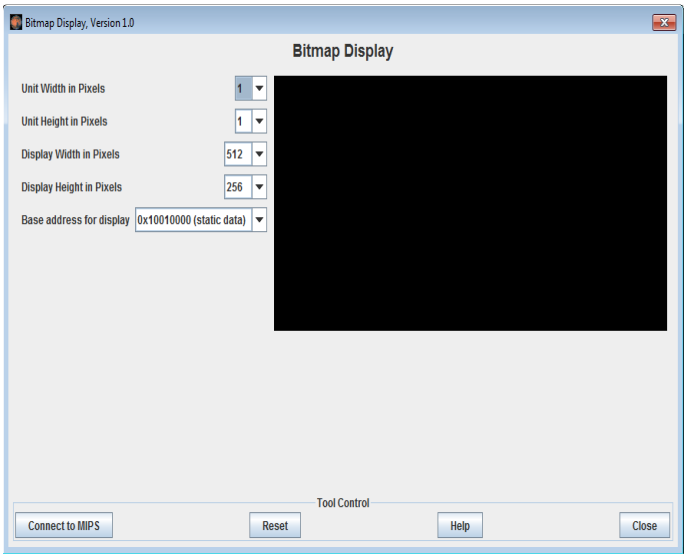

Figure 3: Bitmap Display

IV.MASTER SNAKE GAME

Algorithm to implement master snake game :

Data segment
Step1:Initialize screen colors, score variable and snake information**.**
Step2: Stores how many points are received for eating a fruit and increases as program gets harder
Step3: Speed the snake as it moves, increases as game progresses
Step4:Array to store the scores in which difficulty should increase
Step5: Display end message "You have died.... Your score was: "and replay Message: "Would you like to replay?"

Text segment:
Step1:Initially moving up direction variable,
119 -Moving up – W,
115 - Moving down – S,
97 - Moving left – A,
100 - Moving right – D,
Numbers are selected due to ASCII characters.
Step2:The array stores the screen coordinates of a direction change once the tail hits a position in this array, its direction is changed this is used to have the tail follow the head correctly
Step3: Draw border, initial snake position, pellet and check for direction change
Step4: Update snake head and tail position, check collision and increase difficulty.
Step 5: If collision occurred display the end message if reply is yes continue else exit.

How to simulate
1. Open the MIPS simulator.
2. Load the snake.asm file into MIPS simulator with File -> Open.
3. Go to Run -> Assemble
4. Go to tools -> Bitmap Display

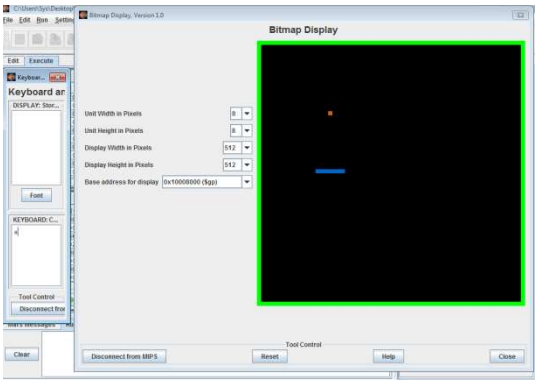

Figure 4: Output of Master Snake Game

5. The Bitmap Display settings should be as follows:
   - Unit Width: 8
   - Unit Height: 8
   - Display Width: 512
   - Display Height: 512
   - Base Address: $gp
6. Go to tools -> Keyboard and Display MMIO Simulator
7. Press connect to MIPS on both of the displays
8. Go to Run -> Go
9. All controls should take place in the lower portion of the Keyboard and Display Simulator

Controls

- W-up
- S-down
- A-left
- D-right

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. Interfacing of keyboard and master snake game is designed and developed using MIPS simulator and results are shown in figure 2 and figure 4.

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.
3. Brorsson, M., MIPS - A Simulation and Development Environment Using Animation for Computer Architecture Education, Workshop on Computer Architecture Education, 29th International Symposium on Computer Architecture, Anchorage AK, 2014.
4. Downcast Systems, MIPS 2.0, http://www.downcastsystems.com/MIPS ter/, retrieved 21 November 2005
5. J.Garton. ProcessorSim - A visual MIPS R2000 processor simulator. http://jamesgart.com/procsim/, 2005
6. Larus, J., SPIM: An MIPS32 simulator, http://www.cs.wisc.edu/~larus/spim.html, retrieved 21 November 2005.
7. 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
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.
9. Sun Microsystems, Java look and feel GraphicsRepository, http://java.sun.com/developer/techDocs/hi/repository /, retrieved 21 November 2005.
10. 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. Todd Austin, Eric Larson, and Dan Ernst. SimpleScalar: An Infrastructure for Computer System Modeling. *IEEE Computer*, 35(2):59–67, February 2002.