Scholars Journal of Engineering and Technology

OPEN ACCESS

Abbreviated Key Title: Sch J Eng Tech
ISSN 2347-9523 (Print) | ISSN 2321-435X (Online)
Journal homepage: https://saspublishers.com

Design of 16 Bit RISC Processor and Implementation using MIPS Technique

Rakesh C R[1*], Chetan S[2], J S Baligar[3]

[1]M. Tech Student, Department of Electronics and Communication, Dr. Ambedkar Institute of Technology, Bengaluru-560056
[2]Assistant Professor, Department of Electronics and Communication, Dr. Ambedkar Institute of Technology, Bengaluru-560056
[3]Associate Professor, Department of Electronics and Communication, Dr. Ambedkar Institute of Technology, Bengaluru-560056

DOI: 10.36347/sjet.2023.v11i11.002 | **Received:** 08.10.2023 | **Accepted:** 17.11.2023 | **Published:** 20.11.2023

***Corresponding author:** Rakesh C R
M. Tech Student, Department of Electronics and Communication, Dr. Ambedkar Institute of Technology, Bengaluru-560056

Abstract **Original Research Article**

This research paper presents design & simulation of a high performance five stage pipelined 8 bit or 16-bit Microprocessor without Interlocked Pipeline Stages (MIPS), which is a Reduced Instruction Set Computing (RISC) architecture based processor. The purpose of RISC microprocessor is to execute a minuscule batch of instructions, with the intention of proliferating the celerity of the processor. This processor was designed with 5 phases of pipeline in particular Instruction Fetch (IF), Instruction Decode & Register Fetch (ID), Execution & Address Calculation (EX), Memory Access (MEM) and Write Back (WB) modules. The designing process was done using a myriad of modules which are the ALU, Control Unit, Program Counter, MUX, Instruction Memory, Data Memory, CPU, Register File, and Sign Extension. The Proposed design is developed by Verilog HDL and Simulated by Modelsim 6.4 c and Synthesized by Xilinx tool and proposed system implemented in FPGA Spartan 3 XC3S 200 TQ-144.

Keywords: RISC microprocessor, Instruction Memory, Instruction Fetch (IF), MIPS architecture.

INTRODUCTION

Microprocessors and microcontrollers are typically constructed in the proximity of two major computer architectures: Complex Instruction Set Computing (CISC) and Reduced Instruction Set Computing (RISC). The CISC concept is on the basis of the Instruction Set Architecture (ISA) architecture, which redoubles performance with numerous instructions utilizing a variable number of operands and an out spread variation of addressing modes in various locations in its Instruction Set. As a result, they have different execution times and lengths, requiring a sophisticated Control Unit that occupies a massively large place on the chip. RISC processors often support a subset of instructions supported by their CISC counterparts. A display that compares how many instructions there are in a RISC CPU to that of a CISC processor.

The MIPS processor is the one we'll be thinking about in this session. Researchers at Stanford University created the RISC (Reduced Instruction Set Computer) processor called as the MIPS in 1984. RISC processors often support fewer, much simpler instructions than their CISC (Complex Instruction Set Computer) counterparts (such as the Intel Pentium CPUs).

MIPS (Microprocessor without Interlocked Pipelined Stages) is a RISC instruction set architecture (ISA) that was created by MIPS Computer Systems, an American business that is currently known as MIPS Technologies: A:1-19.

Several generations of MIPS, namely MIPS I, II, III, IV, and V. Additionally, there are five MIPS32/64 versions (for implementations that are 32 or 64 bits, respectively). 64-bit equivalents were later developed; the initial MIPS designs were limited to 32-bit architectures. The most recent version of MIPS is MIPS32/64 Release 6, which was released in April 2017. What separates MIPS32/64 from the main difference between MIPS I-V and its user mode architecture is the privileged kernel mode System Control Coprocessor.

The MIPS architecture has multiple potential extensions. MIPS-3D is a simple collection of floating-point SIMD instructions; MDMX (MaDMaX), a more comprehensive integer SIMD instruction set using 64-bit floating-point registers; and MIPS-16e, which adds compression to the instruction stream to let programs take up less memory, handle common 3D operations.

In computer architecture classes at technical institutions and universities, the MIPS architecture is commonly studied.[10] The design had a big impact on later RISC systems like Alpha.

In embedded systems such as routers and home gateways, MIPS processors were in use as of April 2017. General-purpose computing was MIPS's original intended use. The 1980s and 1990s saw the widespread use of MIPS processors for personal, workstation, and server computers by a number of companies, including Silicon Graphics, Tandem Computers, NEC, Pyramid Technology, SiCortex, Siemens Nixdorf, and Digital Equipment Corporation. The Nintendo 64, Sony PlayStation, PlayStation 2, and PlayStation Portable were among the gaming consoles that once used MIPS CPUs. Nineties supercomputers also made extensive use of MIPS processors, even though none of these systems are on the TOP500 today.

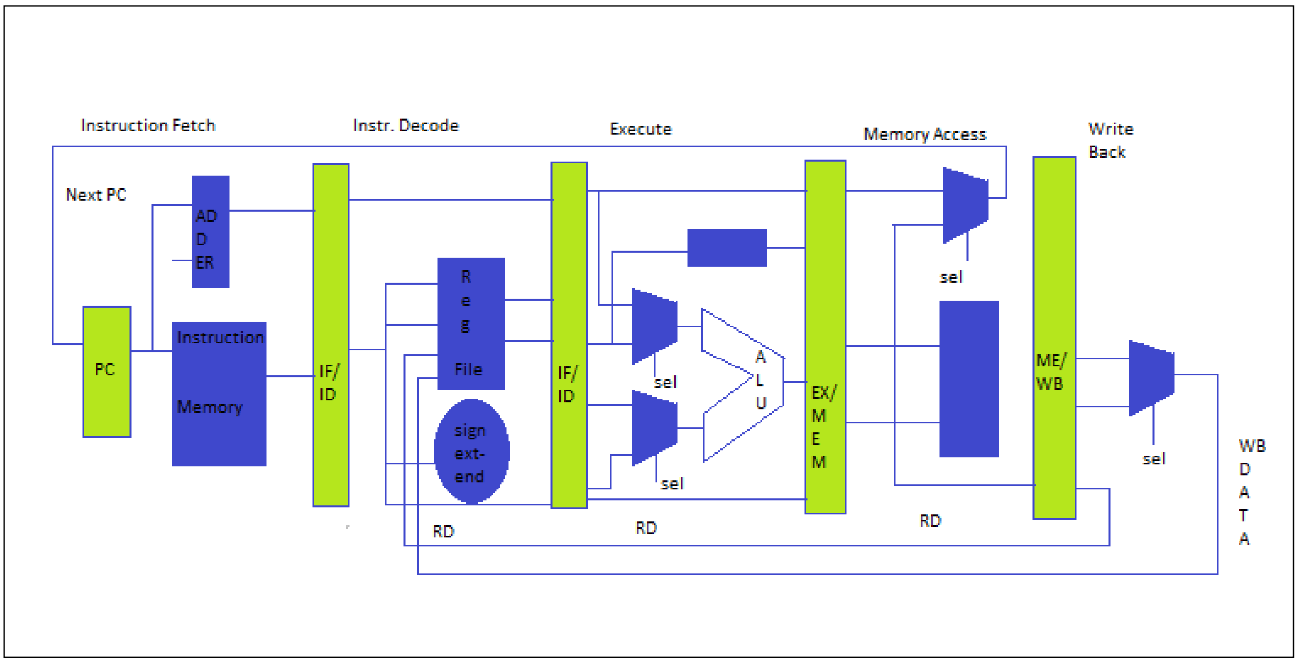
The idea is that a RISC processor can be produced significantly faster than a CISC processor due to its more straightforward design. Nowadays, it is widely acknowledged that RISC processors function better than CISC processors, and even Intel Pentium, the only widely used CISC processor still in use, internally converts CISC instructions into RISC instructions before they are performed.

The architecture of RISC processors is typically load-store. Consequently, there are two instructions for gaining access to memory: a load (l) instruction for obtaining information from memory, and a store (s) instruction for storing information there. Additionally, it implies that none of the other commands can directly access memory. Consequently, a CISC instruction like "add this byte from memory to register 1"

PROPOSED SYSTEM

The suggested 16 bit RISC Processor's design modeling includes an ALU, a control unit, a shifter, and a barrel shifter rotator. The CPU is designed with the Harvard architecture and can distinguish between program memory and data memory. The suggested design employs a two-stage pipeline in both the positive and negative edges, reducing latency while increasing speed. During the first stage of CPU development, a total of 33 instructions/operations are designed. The two-stage pipelining performs four operations: fetch, decode, execute, and read/write back. The data and instructions are drawn from memory in Fetch. When it comes to a decoder, the data drawn from memory are separated by activating three procedures.

BLOCK DIAGRAM:



BLOCK DIAGRAM EXPLANATION:

The project includes phases for instruction fetch (IF), decode (ID), execution (EX), data memory (MEM), and write back (WB). A separate block is intended for jump instruction. The pipeline is made up of overlapping sets of instructions. The pipeline method shortens the execution time of instructions. A single instruction is executed by the processor in a single cycle. Each instruction either goes through all phases or goes according to the instructions. When a program counter is loaded with an address, it operates as a pointer to program memory.

CONTROL UNIT:

Control unit has two operational components. In our suggested architecture, its serves hub for all necessary processing component coordination, generating the required control signals. It is in charge of producing the signals that users choose to use

FETCH UNIT:

Instruction memory: The fetch33 bit of Read the directions on this memory, that possesses the instructions that the processor will actually execute. The address where the directive will be fetched from the instruction memory during the following clock cycle is contained in program counter. At the start of every clock cycle, the computer is increased by one. The fetch unit is at work on the cycle's positive edge.

DECODE UNIT:

The three operations ALU, universal shifter, and shifter rotator are carried out in the decoder, which has two addressing modes as input: immediate addressing mode or a register addressing mode. The control unit has two decoders; the first performs mathematical and logical operations, and the second decoder rotates and shifts data. Decoding is carried out in a negative cycle.

EXECUTE UNIT:

The actual computation takes place throughout the execution phase. The specified control signals cause all of procedures to be carried out. The control signal is precisely followed to operate the ALU, shifter, and shifter rotator. The cycle's positive clock edge used for this action.

WRITE BACK UNIT:

At this point, both single-cycle and two-cycle directives are utilized to write their result into the register file.

ALU:

The ALU design uses 8 operations and has two units: one for logical operations (such AND, NAND, OR, and XOR) and one for arithmetic operations. CSA The functionalities of adder, subtractor, efficient multiplier, and divider are used.

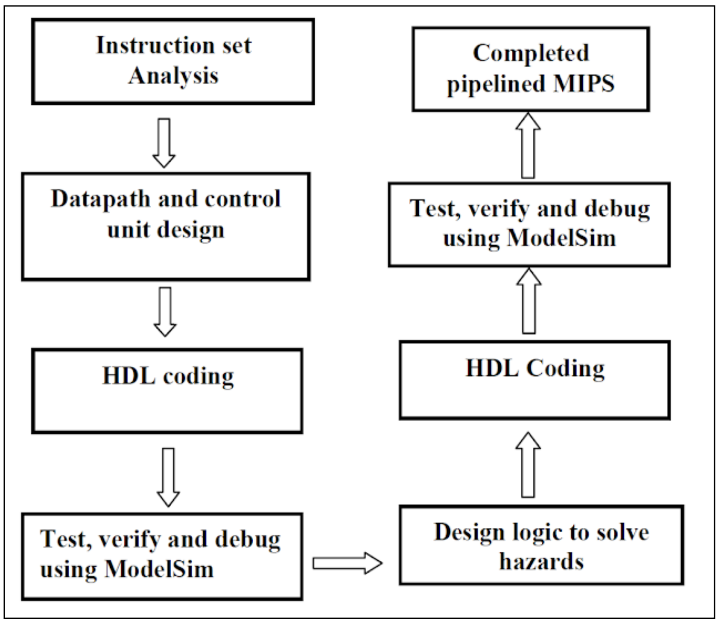
SHIFTER:

The shifter is executing the right shift operation and the left shift operation, according to the universal shift register.

REGISTERS:

A register is type of memory that has the capacity to hold multiple bits of data. Its often implemented as a collection of flip-flops that share control signals to regulate the transfer of data to and from the register.

PROCESS DIAGRAM:


Instruction set Analysis
Datapath and control unit design
HDL coding
Test, verify and debug using ModelSim
Design logic to solve hazards
HDL Coding
Test, verify and debug using ModelSim
Completed pipelined MIPS

SOFTWARE PREREQUISITE
VERIFICATION TOOL

- Modelsim 6.4a

SYNTHESIS TOOL

Xilinx ISE 9.1/ Xilinx 13.2

MODELSIM
Enhanced Simulation and Debugging Capabilities with Modelsim SE

Our debug and simulation environment, ModelSim SE, combinesthe quickest and most reliable graphical user interface (GUI) with high speed and is available for Windows, Linux, and UNIX operating systems.

What Has Model Sim SE Added?

- Improvements to the FSM debug options, such as control over the transitional table and alerts. Coverage of FSM Multi-state Transitions has been Added (i.e., coverage for all potential FSM state sequences).
- Better debugging using hyperlinked navigation between source files viewed and between objects and their specification.
- Data flow window may now calculate and show all connections between nets.
- Improved data management for code coverage with fine-grained control More details are available in the source window.
- The System Verilog types real, multidimensional arrays, packed unions, and structures with fixed sizes are now supported by toggle coverage.

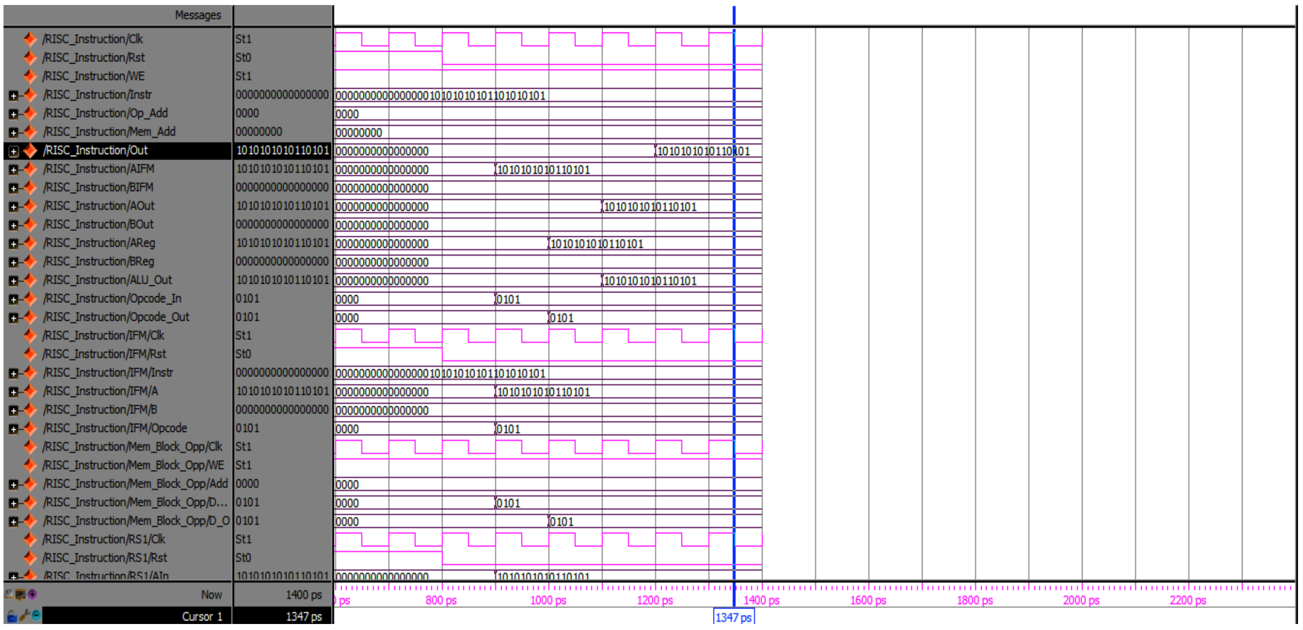
XILINX ISE INTRODUCTION

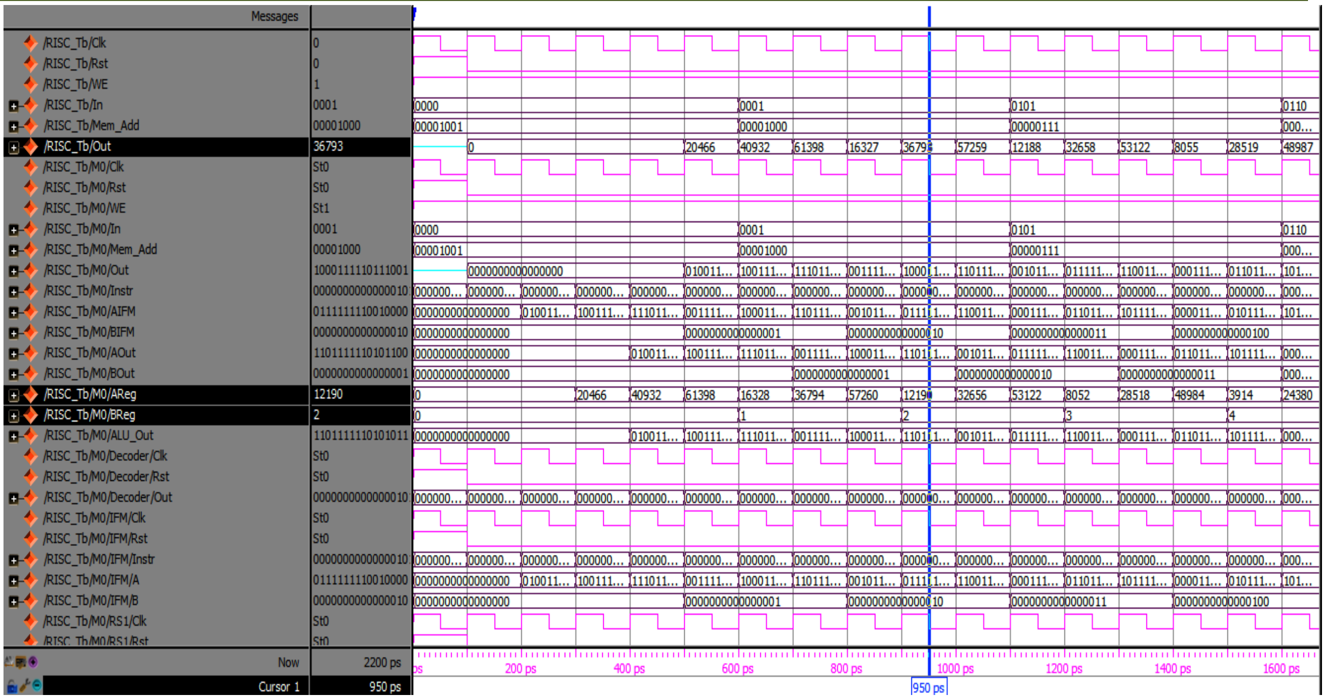
The previous 25 years have seen Xilinx led that which is programmable revolution with the development and ongoing enhancement of FPGA platform technology. Afterwards, with the FPGA, evolved from a highly flexible SASP and ASIC replacement for multiple markets and applications, starting as a prototype and glue logic tool. For our clients and Xilinx, it's the "programmable imperative", replacing the programmable revolution of the past. Xilinx® FPGAs are strategically essential to top-notch system companies aiming to thrive and remain competitive in these extremely unstable times for the global economy.

Imperative with Programming

To ensure that succeed, consumers must capable of achieve more with less, reduce risk whenever feasible, and distinct from the throng. It's called as the imperative of programming. It basically entails seeking to simultaneously satisfy conflicting demands caused by evolving business challenges (i.e., capped spending on engineering, rising One-time engineering expenses for ASIC and ASSP, spiraling intricacy, and elevated risk) and evolving product specifications (i.e., cost, power, performance, and density). The requirement that programming denotes a twofold dedication to Xilinx. First, programmable silicon innovations that deliver market-leading amount for every Among the five main criteria (power, price, performance, features, and density) at every manufacturing node should continue to be developed. The subsequent pledge is to offer customers with what Xilinx calls referred Known as "targeted design platforms," design platforms that are simpler, more intelligent, and more strategically viable for developing top-tier FPGA-based solutions across various industries.

SNAPSHOTS


```
Messages
/RISC_Instruction/Clk                    St1
/RISC_Instruction/Rst                    St0
/RISC_Instruction/WE                     St1
/RISC_Instruction/Instr                  0000000000000000
/RISC_Instruction/Op_Add                 0000
/RISC_Instruction/Mem_Add                00000000
/RISC_Instruction/Out                    1010101010110101
/RISC_Instruction/AIFM                   1010101010110101
/RISC_Instruction/BIFM                   0000000000000000
/RISC_Instruction/AOut                   1010101010110101
/RISC_Instruction/BOut                   0000000000000000
/RISC_Instruction/AReg                   1010101010110101
/RISC_Instruction/BReg                   0000000000000000
/RISC_Instruction/ALU_Out                1010101010110101
/RISC_Instruction/Opcode_In              0101
/RISC_Instruction/Opcode_Out             0101
/RISC_Instruction/IFM/Clk                St1
/RISC_Instruction/IFM/Rst                St0
/RISC_Instruction/IFM/Instr              0000000000000000
/RISC_Instruction/IFM/A                  1010101010110101
/RISC_Instruction/IFM/B                  0000000000000000
/RISC_Instruction/IFM/Opcode             0101
/RISC_Instruction/Mem_Block_Opp/Clk      St1
/RISC_Instruction/Mem_Block_Opp/WE       St1
/RISC_Instruction/Mem_Block_Opp/Add      0000
/RISC_Instruction/Mem_Block_Opp/D...     0101
/RISC_Instruction/Mem_Block_Opp/D_O      0101
/RISC_Instruction/RS1/Clk                St1
/RISC_Instruction/RS1/Rst                St0
/RISC_Instruction/RS1/AIn                1010101010110101
Now                                      1400 ps
Cursor 1                                 1347 ps
```

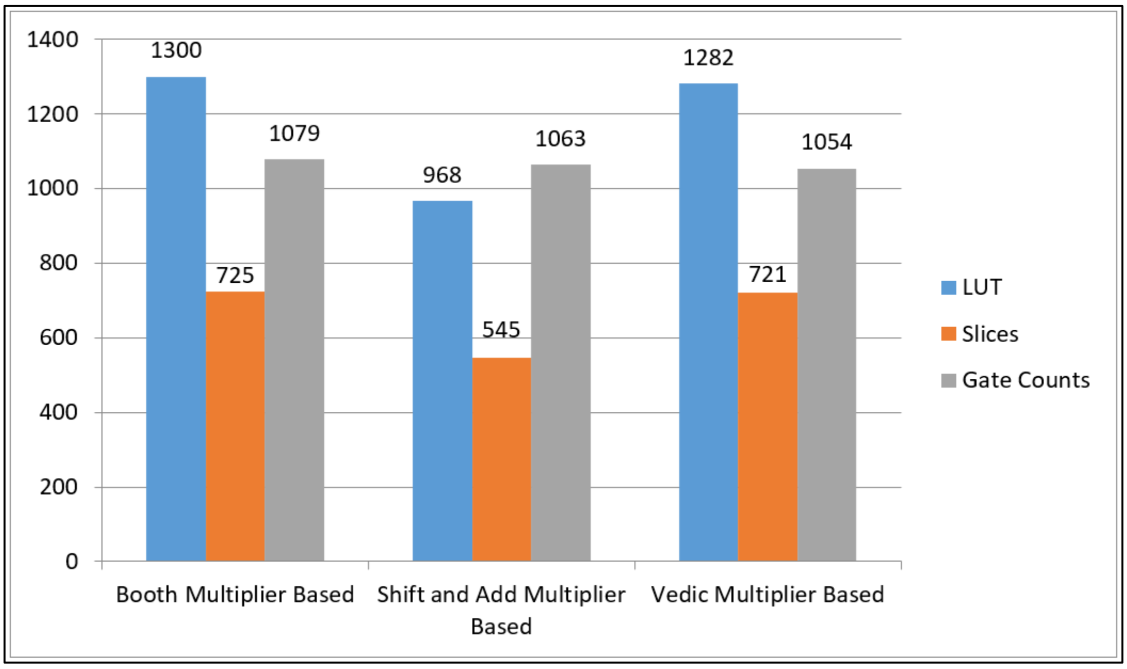


```
Messages
/RISC_Tb/Clk                 0
/RISC_Tb/Rst                 0
/RISC_Tb/WE                  1
/RISC_Tb/In                  0001
/RISC_Tb/Mem_Add             00001000
/RISC_Tb/Out                 36793
/RISC_Tb/M0/Clk              St0
/RISC_Tb/M0/Rst              St0
/RISC_Tb/M0/WE               St1
/RISC_Tb/M0/In               0001
/RISC_Tb/M0/Mem_Add          00001000
/RISC_Tb/M0/Out              1000111110111001
/RISC_Tb/M0/Instr            000000000000010
/RISC_Tb/M0/AIFM             0111111110010000
/RISC_Tb/M0/BIFM             0000000000000010
/RISC_Tb/M0/AOut             1101111110101100
/RISC_Tb/M0/BOut             0000000000000001
/RISC_Tb/M0/AReg             12190
/RISC_Tb/M0/BReg             2
/RISC_Tb/M0/ALU_Out          1101111110101011
/RISC_Tb/M0/Decoder/Clk      St0
/RISC_Tb/M0/Decoder/Rst      St0
/RISC_Tb/M0/Decoder/Out      000000000000010
/RISC_Tb/M0/IFM/Clk          St0
/RISC_Tb/M0/IFM/Rst          St0
/RISC_Tb/M0/IFM/Instr        000000000000010
/RISC_Tb/M0/IFM/A            0111111110010000
/RISC_Tb/M0/IFM/B            0000000000000010
/RISC_Tb/M0/RS1/Clk          St0
/RISC_Tb/M0/RS1/Rst          St0
Now                          2200 ps
Cursor 1                     950 ps
```

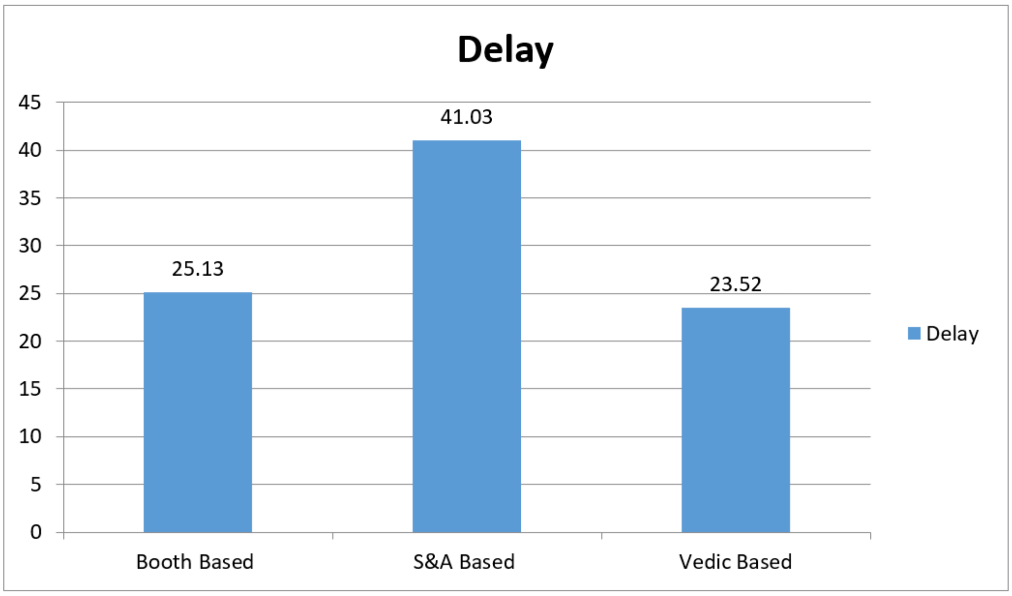
SYNTHESIS REPORT

Method Name	Area in Number of LUT			Delay
Spartan 3 XC 3S 4000-4FG1156	**LUT**	**Slices**	**Gates**	**Delay**
RISC Processor based On Booth Multiplier	1300	725	107994	25.131
RISC Processor based On S&A	968	545	105476	41.035 ns
RISC Processor based On Vedic Multiplier	1282	721	106388	23.527 ns

AREA:



DELAY:



CONCLUSION

This study describes how pipelining can be used to effectively execute a 16-bit Microprocessor without Interlocked Pipeline Stages (MIPS) based RISC processor. Each direction in a five step pipelining system is executed once every clock cycle. This design shows how to use a MIPS-based CPU that is capable of handling various Register type, Jump type, and instantaneous type of instructions, each of which has a unique configuration.

REFERENCES OR BIBLIOGRAPHY

1. http://en.wikipedia.org/wiki/MIPS_architecture.
2. http://logos.cs.uic.edu/366/notes/mips%20quick%20tutorial.htm
3. David A. Patterson and John L. Hennessy, Computer Organization and Design, 3rd ed.
4. Lecture notes by Howard Huang, University of Illinois at Urbana-Champaign. [Online]. Available: http://www.howardhuang.us/teaching/cs232/11-Single-cycle-MIPS-processor.pdf
5. Lecture notes by Howard Huang, University of Illinois at Urbana-Champaign. [Online]. Available: http://www.howardhuang.us/teaching/cs232/12-Multicycle-datapath.pdf
6. Lecture notes by Howard Huang, University of Illinois at Urbana-Champaign. [Online]. Available: http://www.howardhuang.us/teaching/cs232/15-Pipelining.pdf
7. MIPS Official Website. Available: http://www.mips.com/products/architectures/mips32/