Volume 3, Issue 2, February 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Performance Improvement of MIPS Architecture by Adding New Features

Galani Tina
Department of Electrical
VJTI College of Engg, Mumbai India

R.D.Daruwala
Department of Electrical
VJTI College of Engg, Mumbai India

Abstract - RISC or Reduced Instruction Set Computer is a design philosophy that has become a mainstream in Scientific and engineering applications. Increasing performance and gate capacity of recent FPGA devices permits complex logic systems to be implemented on a single programmable device. This paper targets to develop a 32- bit MIPS RISC processor architecture in VHDL language that detects the pipeline hazards during the multithreading and reduce Cycle per instruction (CPI) by eliminating the pipeline hazards. The module functionality and performance issue like area, power dissipation and propagation delay are analysed at 90nm process technology using Virtex4 XC4VLX15 XILINX tool.

Keywords: MIPS Processor, Reduced Instruction Set Computer (RISC), VHDL, Pipeline, Xilinx 12.1, FPGA

I. INTRODUCTION

Nowadays, computersand mobile phones is indispensable tool for most of everyday activities. This places an increasing burden on the embedded microprocessor to provide high performance while retaining low power consumption and small die size, which increase the complexity of the device [3].

However, as products grow in complexity more processing power is required while the expectation on battery life also increases. With the rapid development of silicon technology RISC includes extensions to RISC concepts that help achieve given levels of performance at significantly lower cost than other systems. The main features of RISC processor are the instruction set can be hardwired [7] to speed instruction execution. With reconfigurable devices such as FPGAs based core design any set of task can be configured. It sustains any system level change without costly hardware replacement and thus the design process is very fast and cost effective.

In the present work, the design of modules to remove pipeline hazards along with MIPS processor architecture is presented. It has a complete instruction set, instruction and data memories, 32 general purpose registers, Arithmetical Logical Unit (ALU) for basic operation, forwarding unit, hazard detection unit and flushing unit. The instruction cycle consists of five stages namely fetch, decode, execute, memory access and write back. Control unit determines the types of instruction to execute. The remainder of this paper is organized as follows. Section II explains types of hazards and design modules to remove hazards. Section III presents the implementation of modules. Section IV presents the simulation results implemented in advanced 90nm process technology. Section V discusses summary with the implementation of the MIPS RISC design topology. The final section presents the Conclusion and References.

II. BACKROUND

Pipelining, a standard feature, is an implementation technique used to improve both CPI (Cycle Per Instruction) and overall system performance [1]. Pipelining allows a processor to work on different steps of the instruction at the same time, thus more instruction can be executed in a shorter period of time. Thus in pipelining each module of MIPS processor does not wait for the previous instruction to finish before it can execute. Pipelining the MIPS processor introduces events called hazards [1], which prevents the next instruction in the instruction stream from being executing during its designated clock cycle and reduce the speed of the processor. The types of hazards include structural, data and control hazards.

- **Structural hazards** arise when the flow of instructions requires more hardware resources than those available on the platform [1].
- **Data hazards** arise when there is a data dependency between the current instruction and the previous instruction in the pipeline.
- **Control hazards** arise when there is a change in the flow of the program (branch instruction that changes the PC).

A.) Structural hazards: Structural hazards arise when the hardware cannot support the combination of instruction that we want to execute in the same clock cycle. Fortunately, the MIPS instruction set is designed to be pipelined, making it fairly easy for designers to avoid structural hazards when designing a pipeline. However, if the MIPS

processor had been designedwith one memory to be shared between both instruction and data instead of two memories (Instruction memory and Data memory), then structural hazard would occur.

B.) Data hazards: Data hazards arise when an instruction depends on the result of a previous instruction in a way that is exposed by the overlapping of the instructions in the pipeline [1], thus causing the pipeline to stall until the results are made available. One solution to this type of data hazard is called forwarding, which supplies the resulting operand to the dependant instruction as soon it has been computed.

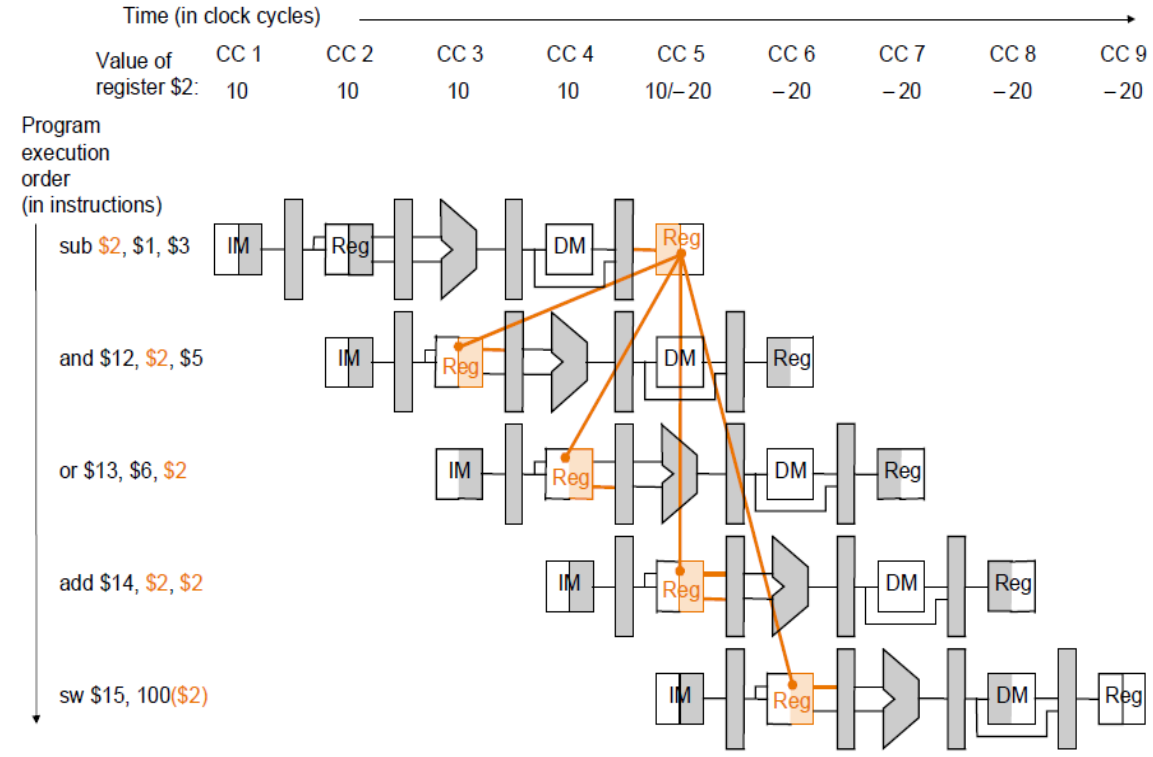

Fig. 1 Pipelined Data Dependencies

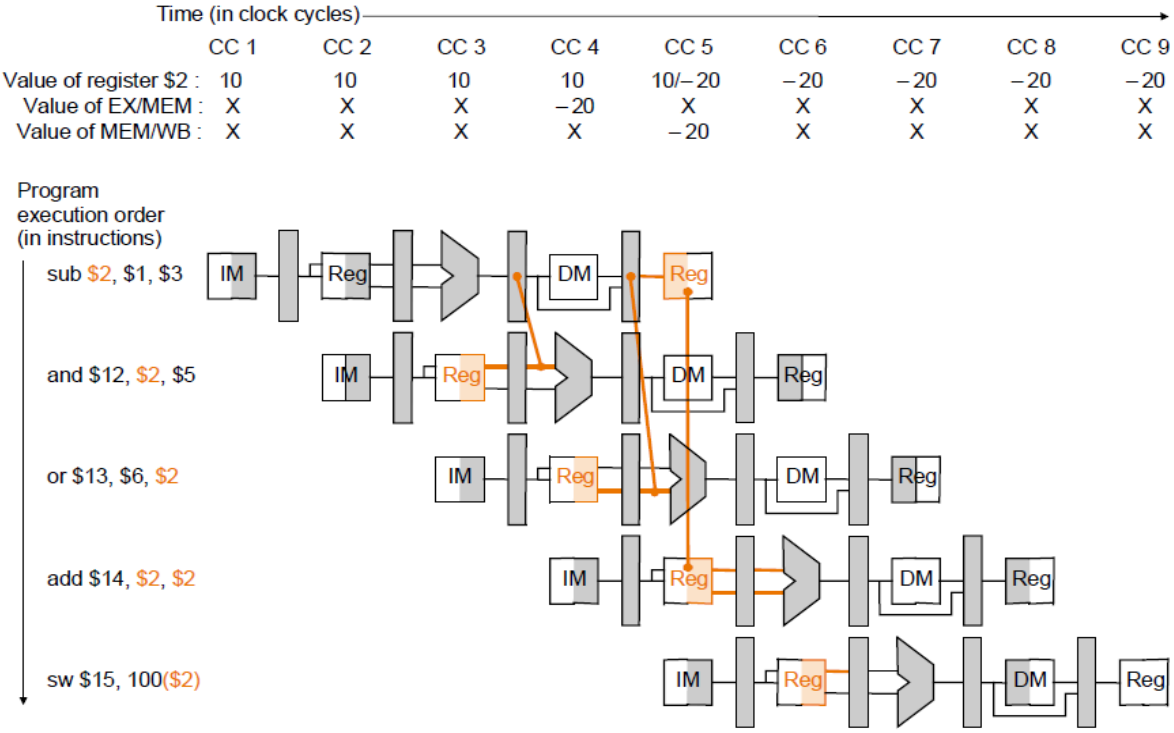

Fig. 2 Pipelined Data Dependencies Resolved with Forwarding

While forwarding is an exceptional solution to data hazards it does not resolve all of them [1]. One instance is when an instruction attempts to read a register value that is going to be supplied by a previous load instruction that writes the same register, called a load-use hazard. At the same time the load instruction is reading data from memory, the subsequent

instruction executing in the execution stage with the wrong data value. The only solution here is to stall the pipeline and wait for the correct data value being used as an operand. In order to detect such hazards, MIPS introduces a Hazard Detection Unit [4] during the instruction decode stage so that it can stall the pipeline between a load instruction and the immediate instruction attempting to use the same register.

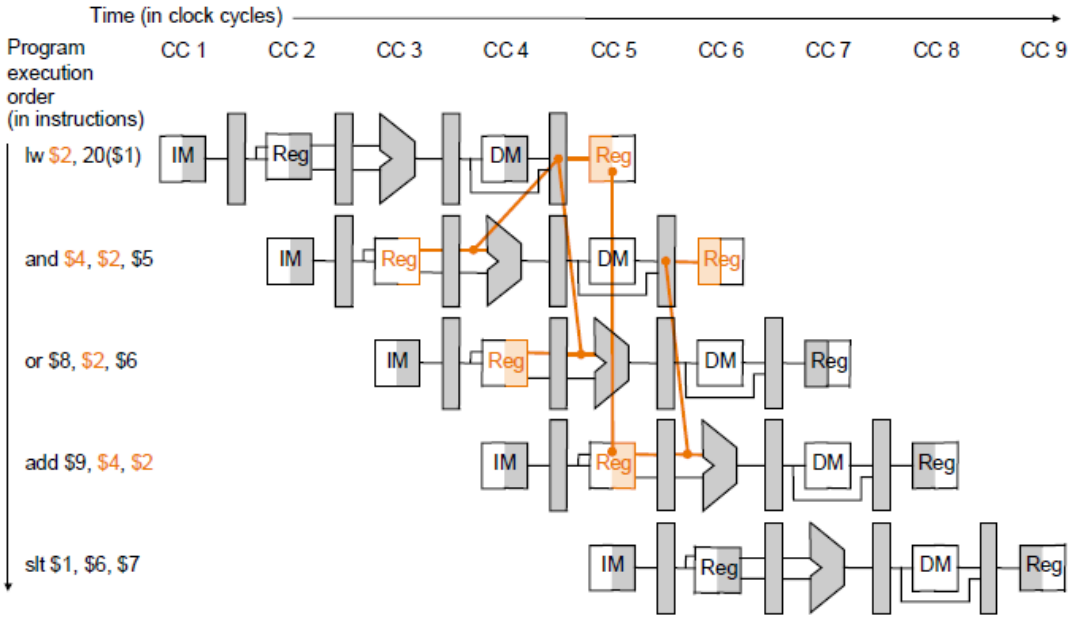

Fig. 3 Pipelined Data Dependencies Requiring Stall

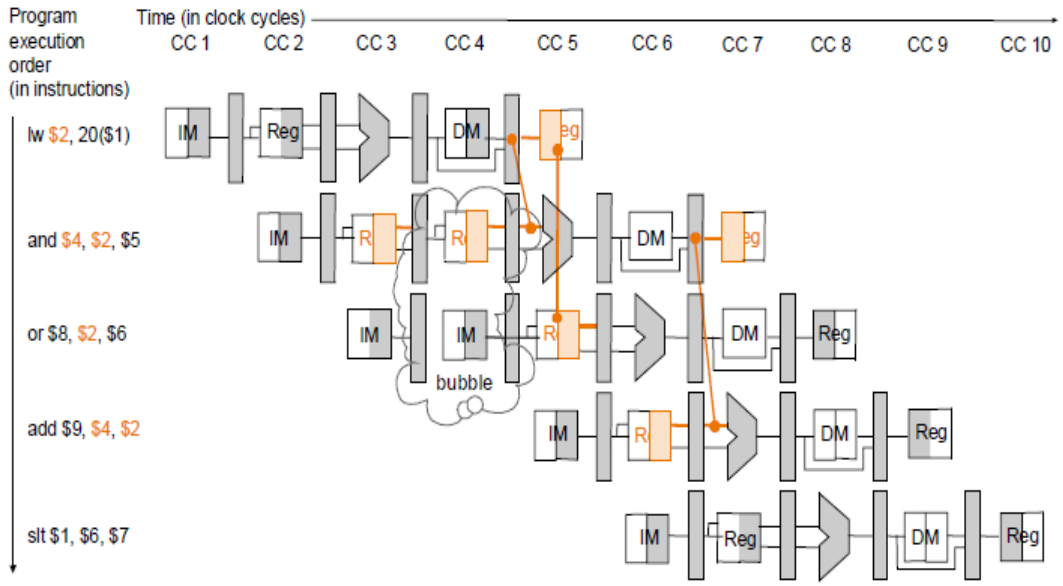

Fig. 4 Pipelined Data Dependencies Resolved with Stall

C.) Control hazard: The last type of hazard is a control hazard also known as a branch hazard. These hazards occur when there is a need to make a decision based on the results of one instruction while other instructions continue executing [1]. For example, a branch on equal instruction will branch to a non-sequential part of the instruction memory if the two register values compared are equal. While the register values are compared, other instructions continue to be fetched and decoded. If the branch is taken, the wrong instructions are fetched into the pipeline and must somehow be discarded. Figure 5 shows three instructions that need to be discarded after it is determined the branch instruction will be taken.

A common solution to these hazards is to continue instruction execution as if the branch is not taken. If it is later determined that the branch is taken, the instructions that were fetched and decoded must be discarded which can be achieved by flushing some of the pipeline registers. Flushing means that all values stored in the pipeline registers are discarded or reset. However in order to reduce the branch hazard to 1 clock cycle, the branch decision is moved from thememory pipeline stage to the instruction decode stage. By simply comparing the registers fetch it can be determined if a branch is to be taken or not.

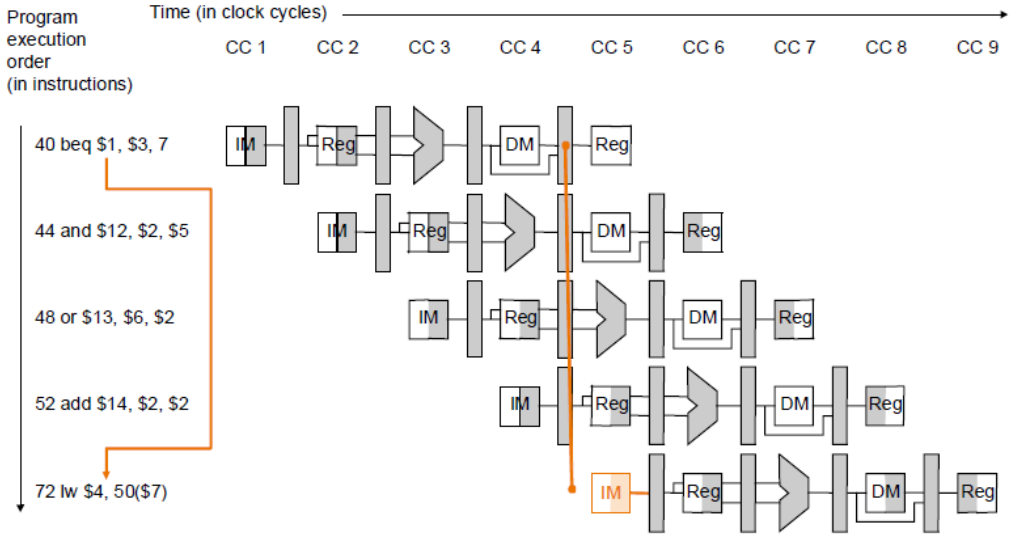

Fig. 5 Pipelined Branch Instructions

III. IMPLEMENTATION

A 32-bit MIPS processor was designed, tested and synthesized. For improving performance processor has following attributes:

(i) Hazard Detection Unit and Correction
(ii) Forwarding Unit
(iii) Flushing Unit to flush IF/ID pipeline register [4]

In implementation for lower CPI first detect a hazard and then forward the proper value to resolve the hazards. The types of hazard condition that can be detected are as following:

Type 1a. EX/MEM.RegisterRd = ID/EX.RegisterRs
Type 1b. EX/MEM.RegisterRd = ID/EX.RegisterRt
Type 2a. MEM/WB.RegisterRd = ID/EX.RegisterRs
Type 2b. MEM/WB.RegisterRd = ID/EX.RegisterRt

Because some instructions do not write registers, this policy is inaccurate; sometimes it would forward when it was unnecessary.

One solution is simply to check to see if the RegWrite signal will be active: examining the WB control filed of the pipeline register during the EX and MEM stages determines if RegWrite signal is asserted [1]. Besides of this check that an instruction in the pipeline has $r0 register as its destination, we want to avoid forwarding because register $r0 is hard wired and always contains the value zero.

The hazard conditions above thus work properly as long as we add EX/MEM RegisterRd != 0 to the first hazard condition and MEM/WB RegisterRd != 0 to the second. Following are the conditions for detecting hazards and control signals for forwarding unit to resolve them:

(1.) EX hazard:
If ((EX/MEM.RegWrite = 1) and (EX/MEM.RegisterRd != 0)and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
ForwardA = 10
If ((EX/MEM.RegWrite =1) and (EX/MEM.RegisterRd != 0)and (EX/MEM.RegisterRd = ID/EX.RegisterRt))
ForwardB = 10
This case forwards the results from the previous instruction to the either input of the ALU. The other case if previous instruction is going to write to the register file and write register number matches the read register number of ALU inputs, then activate the multiplexers to pick the value instead from the pipeline register EX/MEM.

(2.) MEM hazard:
If ((MEM/WB.RegWrite = 1) and (MEM/WB.RegisterRd != 0) and (MEM/WB.RegisterRd = ID/EX.RegisterRs))
Forward A = 01
If ((MEM/WB.RegWrite = 1) and (MEM/WB.RegisterRd!= 0) and (MEM/WB.RegisterRd = ID/EX.RegisterRt))
Forward B = 01

Another potential hazard can occur when there is a conflict between the result of the instruction in the WB stage, in the MEM stage, and the source operand of the instruction in the ALU stage. In this case the result is forwarded from the MEM stage. Thus the control for the MEM hazard would be the following:

If ((MEM/WB.RegWrite = 1) and (MEM/WB.RegisterRd != 0) and (EX/MEM.Register.Rd != ID/EX.RegisterRs) and (MEM/WB.RegisterRd = ID/EX.RegisterRs)) then Forward A = 01

If ((MEM/WB.RegWrite = 1) and (MEM/WB.RegisterRd != 0) and (EX/MEM.Register.Rd != ID/EX.RegisterRt) and (MEM/WB.RegisterRd = ID/EX.RegisterRt)) then Forward B = 01

Hazards occur when we read a value that was just written from memory, as that value won't be available for forwarding unit until the end of memory stage. In order to detect such hazards, MIPS introduces a Hazard Detection Unit during the instruction decode stage so that it can stall the pipeline between load instruction and the immediate instruction attempting to use the same register [1]. The control for the Hazard Detection Unit is this single condition:

If ((ID/EX.MemRead = 1) and ((ID/EX.RegisterRt = IF/ID.RegisterRs)or (ID/EX.RegisterRt = IF/ID.RegisterRt))) then stall the pipeline

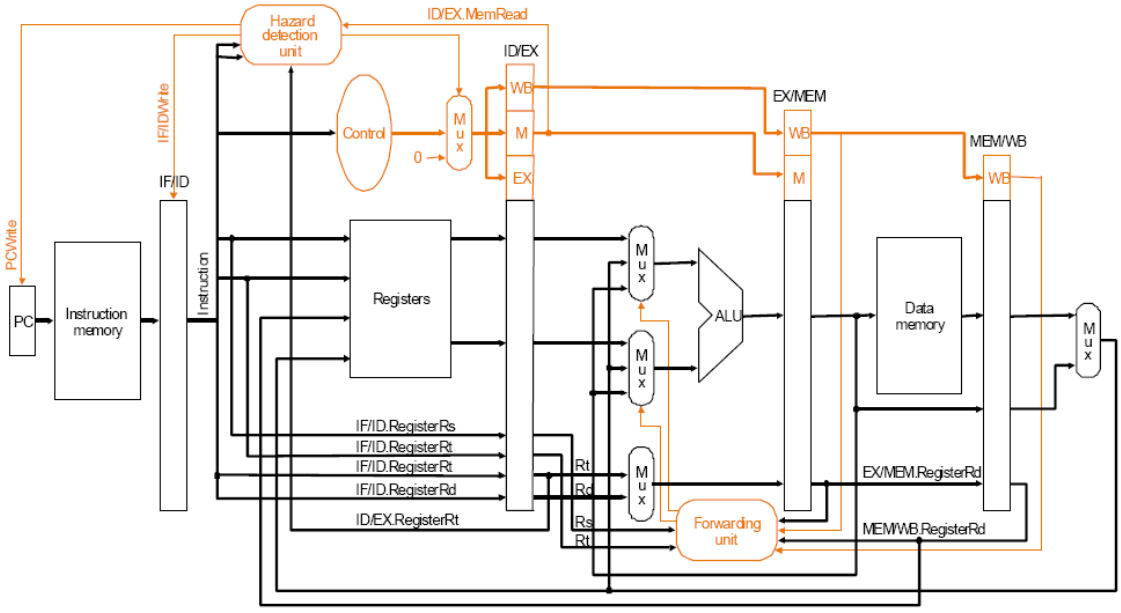

Fig. 6 Pipeline connection for Hazards Detection Unit and Forwarding Unit

One way to improve branch performance is to reduce the cost of the taken branch. If the branch execution moves earlier in the pipeline, then fewer instructions need to be flushed. Thus far next PC assumed for a branch is selected in the MEM stage but if we move branch decision in the ID stage, then only one instruction need to be flushed. For this implementation just move the branch adder from the MEM stage to the ID stage because the PC value and the immediate field in the IF/ID pipeline are already available before ID stage.

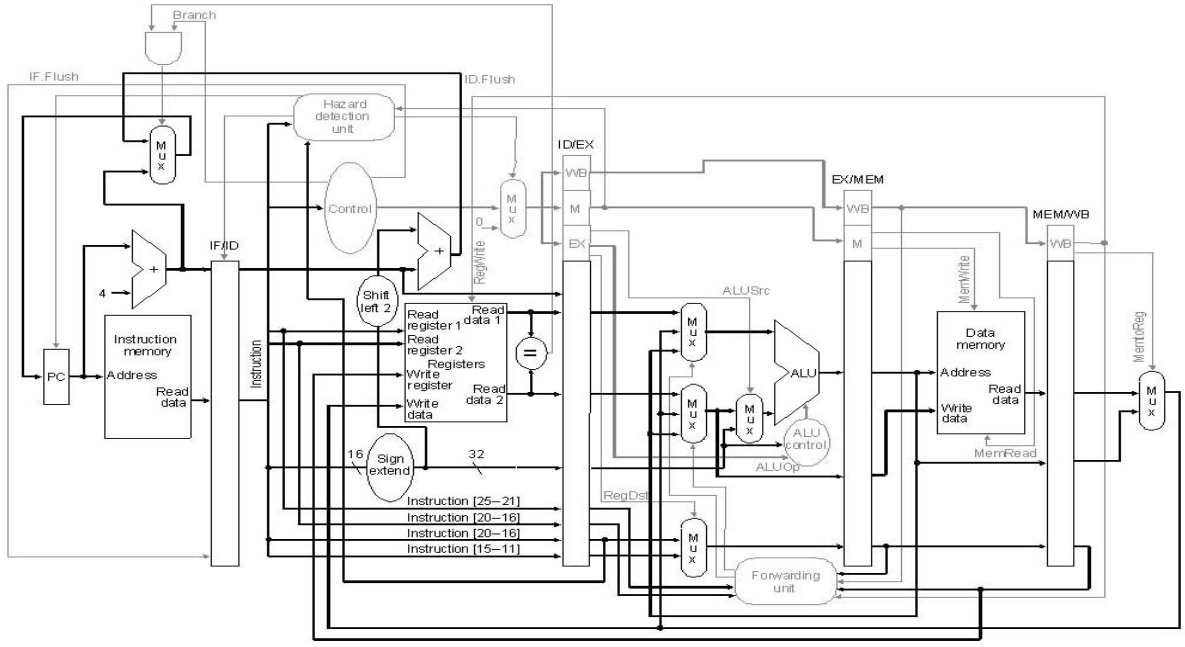

Fig. 7 MIPSPipelined Final Data path and Control

For branch equal, compare the two registers read during the ID stage to see if they are equal. Equality can be tested by first exclusive-ORing their respective bits and then ANDing all the results. By moving the branch execution to the ID stage, there is only one instruction to flush if the branch is taken, the one currently being fetched [1].

To flush instructions in the IF stage, add a control line, called IF.Flush, that zero the instruction field of the IF/ID pipeline register. Clearing the register transforms the fetched instruction into *nop*, an instruction that does no operation to change state.

IV. SIMULATION RESULTS

The performance of the MIPS RISC processor [7] has been evaluated in this research work by using advanced XILINX VIRTEX4 XC4VLX15 technology. The design meets the need of high performance logic solution for high volume, very low cost, consumer – oriented applications.

The synchronization of various operation are done using CLK signal [7]. The MIPS processor is designed with three control signals RD, WR, RESET. If RESET is high then the processor will not perform any operation it will stay in ideal state. CLK is the external clock which is always equal to one which triggers the input and gives us the desired output.

For simulation, a number of instructions from instruction memory fed into the CPU and the outputs of register bank, ALU results and pipeline registers value were monitored. Theinstructions that were tested included register based and immediate, reading and writing datamemory, and a loop that would force the CPU to jump back tothe start of instruction memory and execute those same instructions again. The different adds were important becauseeach exercised different parts of the CPU including the dataforwarding unit, multiple registers and different functionswithin the ALU itself. The jump instruction was very important also in that it exercised the branch detection unit, hazard detection units as well as the ability of the instruction fetch stage to be able to jump to an address and continue execution with only the input of a single stall cycle.

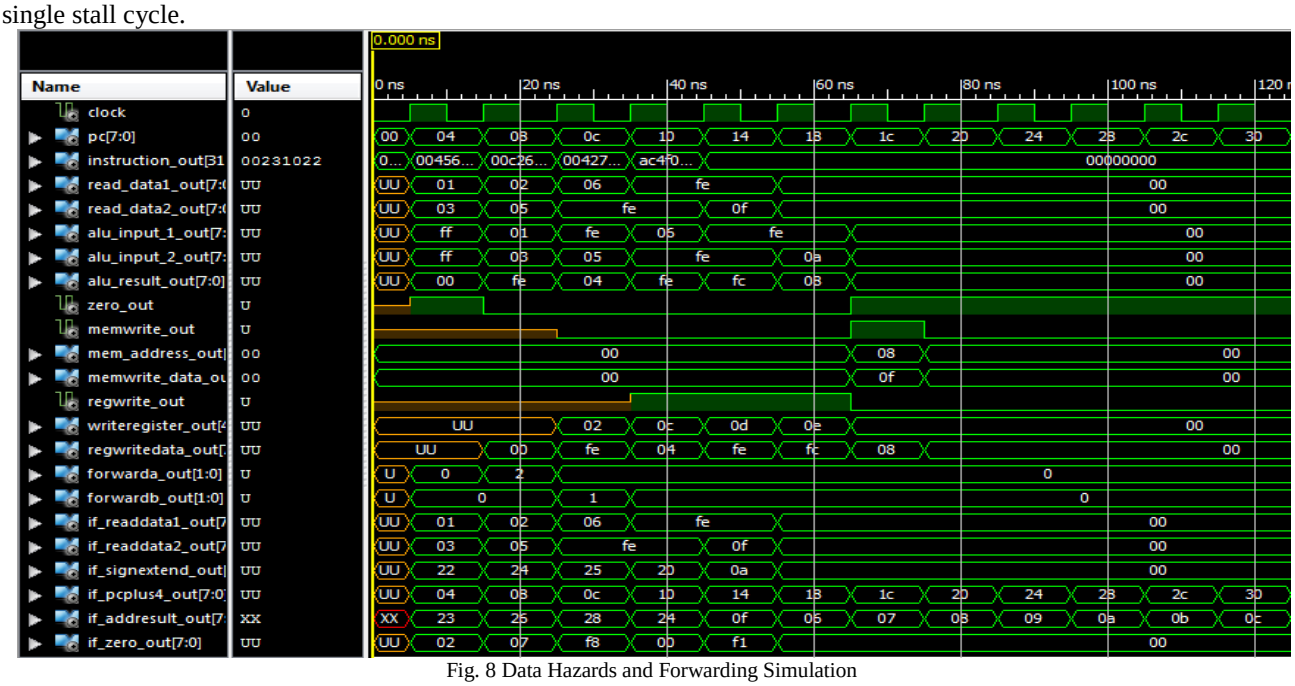

Fig. 8 Data Hazards and Forwarding Simulation

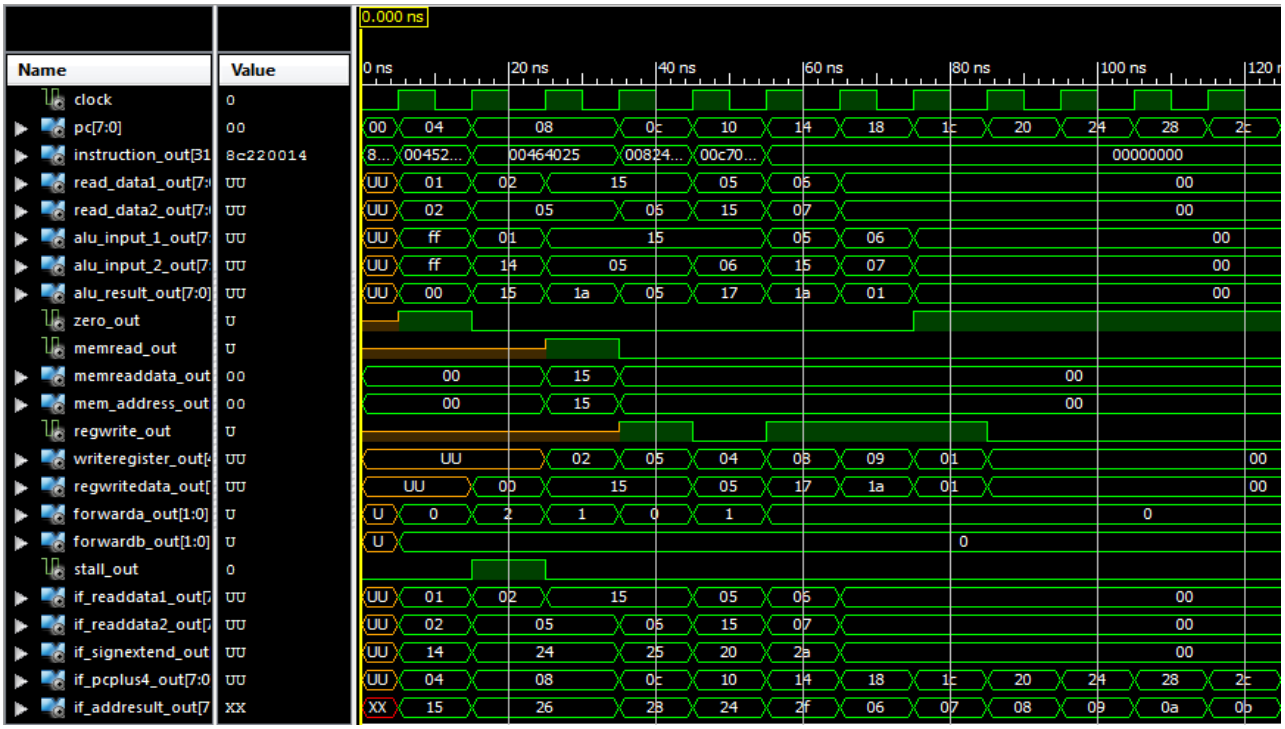

Fig. 9 Data Hazards and Stalling Simulation

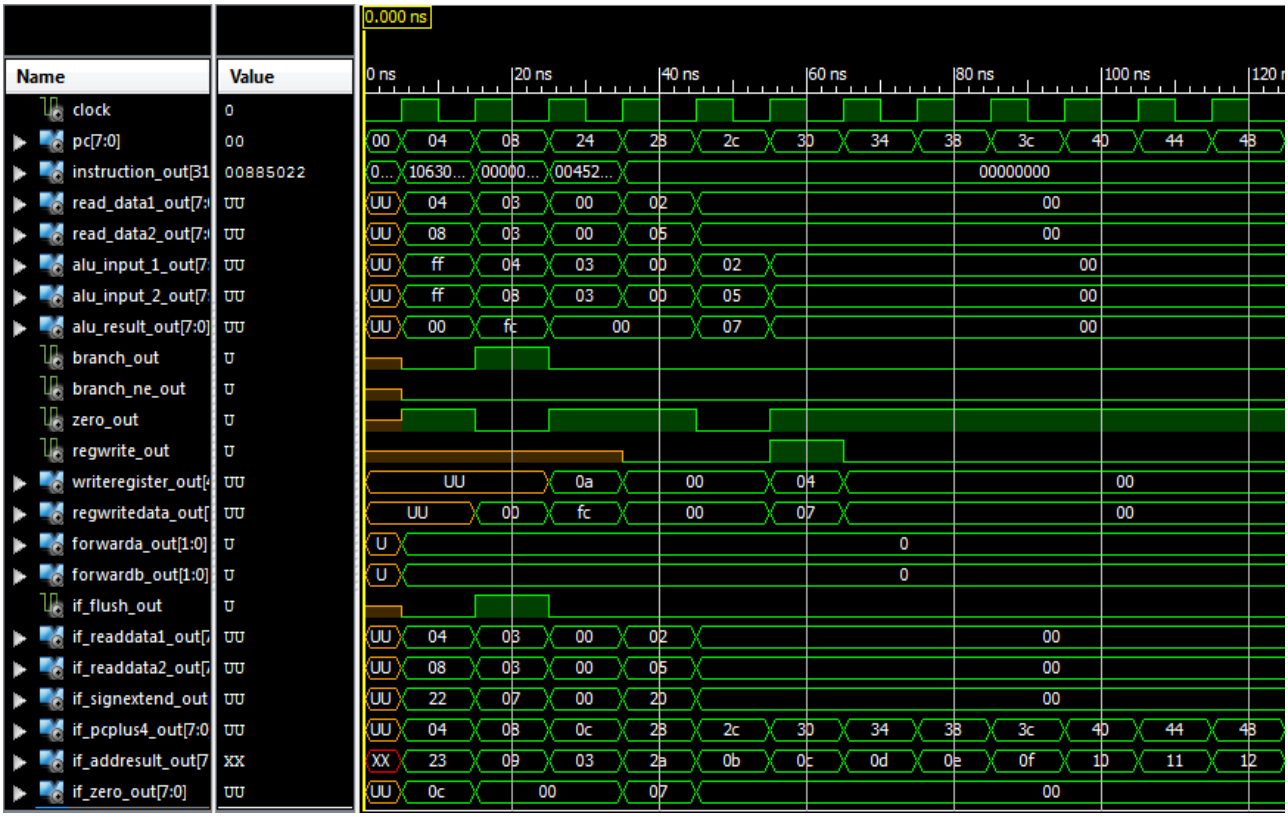

Fig. 10 Branch Hazard Simulation

V. SUMMARY

This section presents the performance of the processor in terms of its Total Power and Delay that are obtained using the Virtex4 XC4VLX15 XILINXtool. Table 1 presents the maximum power dissipation, area occupied and time taken by each module to operate the processor.

TABLE I DELAY, TOTAL POWER AND AREA CALCULATION

Topology	Delay(ns)	Slices Utilized (Area)	AT	AT^2	Power Dissipation 10^{-9}W	Total Power (W)
Instruction Fetch Unit	1.779	49	87.171	155.08	0.159	0.165
Instruction Decode Unit	4.072	290	1180.88	4808.54	0.159	0.168
Control Unit	0.905	15	13.575	12.29	0.159	0.164
Execution Unit	4.495	60	269.7	1212.30	0.159	0.166
Data Memory Unit	3.097	83	257.051	796.09	0.159	0.166
Total	14.348	437	1808.377	6984.3		0.829

It is observed that total delay of the 32-bit MIPS processor is which results after addition of all modules in the design 14.348 ns. The maximum area is occupied by the Decode unit. The overall power dissipation of this processor is observed to be 0.829 W. The top-level design should be partitioned such that the synchronizer blocks are contained in individual modules outside of any functional blocks. This will help to achieve the ideal clock domain scenario (one clock for the entire design) on a block by block basis.By synthesizing the Top Level module of Pipelining design the overall delay results 5.584 ns while by synthesizing the Top Module of Single Cycle Non-Pipelined design the overall delay results 13.613 ns.The following figures show the variation of area, power dissipation and delay of the whole RISC processor and for each sub modules respectively. From the graph it is seen that the maximum power dissipation and chip area is exhibited by the Decode unit.

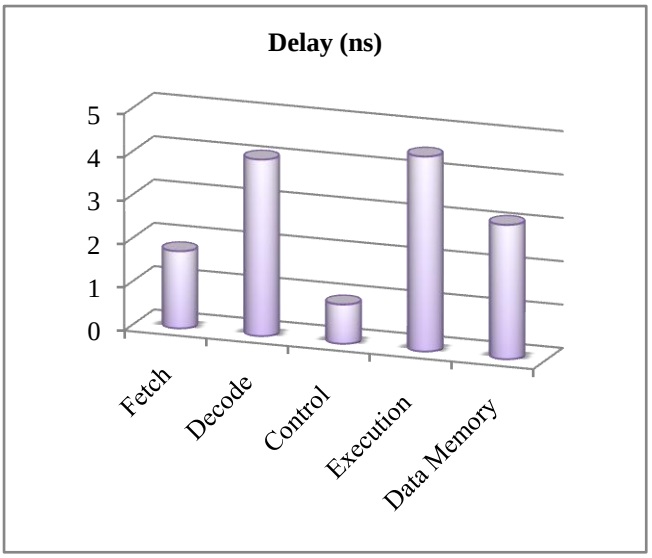

Fig. 11 Delay Estimation

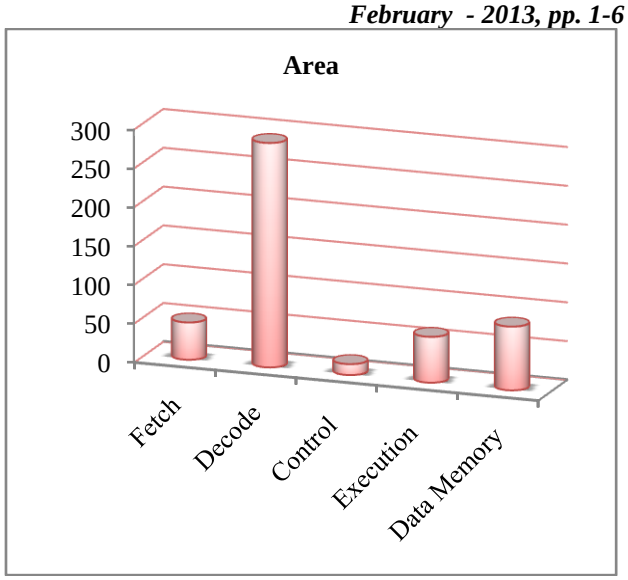

Fig. 12 Area Estimation

VI. CONCLUSION

A 32-bit RISC processor [2] with 16 instruction set has been designed. Every instruction is executed in one clock cycles with 5-stage pipelining. The design is verified through exhaustive simulations. The processor achieves higher performance, lower area and lower power dissipation. The simulation results show that maximum frequency of pipeline processor is increased from73.461 MHz to 179.092MHz.This processor can be used as a systolic core to perform mathematical computations like solving polynomial and differential equations. Apart from this it can be used in portable gaming kits.

REFERENCES

[1] David A. Patterson, John L. Hennessy, "Computer Organization and Design - The Hardware/Software Interface" Second Edition (1998) Morgan Kaufmann Publisher, Inc.

[2] Xiao Li, LongweiJi, Bo Shen, Wenhong Li, Qianling Zhang, "VLSI implementation of a High-performance 32-bit RISC Microprocessor", Communications, Circuits and Systems and West Sino Expositions, IEEE 2002 International Conference on ,Volume 2, 2002 ,pp.1458 – 1461.

[3] KusumlataPisda, DeependraPandey, "Realization & Study of High Performance MIPS RISCProcessor Design Using VHDL", International Journal of Emerging trends in Engineering and Development, Volume 7, Issue 2, November 2012, pp. 134 – 139, ISSN: 2249 – 6149.

[4] Kirat Pal Singh, ShivaniParmar, "VHDL Implementation of a MIPS – 32 bit Pipeline Processor", International Journal of Applied Engineering Research, Volume 7, Issue 11, ISSN:0973 – 4562.

[5] SamiappaSakthikumaran,S.Salivahanan and V.S.Kaanchana Bhaaskaran,"16-Bit RISC Processor Design For Convolution Application",IEEE International Conference on Recent Trends In Information Technology, June 2011, pp.394-397.

[6] RupaliS. Balpande and Rashmi S. Keote, "Design of FPGA based Instruction Fetch & Decode Module of 32-bit RISC (MIPS) Processor, International Conference on Communication Systems and Network Technologies pp. 409 – 413.

[7] R. Uma, "Design and Performance Analysis of 8 – bit RISC Processor using Xilinx Tool", International Journal of Engineering Research and Applications, Volume 2, Issue 2, March – April 2012, pp. 053 – 058, ISSN: 2248 – 9622.

[8] V.N.Sireesha and D.Hari Hara Santosh, "FPGA Implementation of a MIPS RISC Processor", International Journal of Computer Technology and Applications, Volume 3, Issue 3, pp. 1251 – 1253, ISSN: 2229 – 6093.

