Design and Simulation of 32-Bit RISC Architecture Based on MIPS using VHDL

Mr. S. P. Ritpurkar
Department of Electronics and Telecommunication
B. D. College of Engineering, Sevagram, India
email: ritpurkarsagar@gmail.com

Prof. M. N. Thakare
Department of Electronics and Telecommunication
B. D. College of Engineering, Sevagram, India
email: mnt_ent@rediffmail.com

Prof. G. D. Korde
Department of Electronics and Telecommunication
B. D. College of Engineering, Sevagram, India
email: gdkorde@rediffmail.com

***Abstract*—VHDL Very High Speed Integrated Circuits Hardware Description Language) is widely used for ASIC (Application Specific Integrated Circuits) emulation, as well as a solution for applications with high volatility. FPGA (Field Programmable Gate Array) give quick time to market, and its feature of re-programmability often makes them the main part of the system. This paper presents the design of a RISC (Reduced Instruction Set Computer) CPU architecture based on MIPS (Microprocessor Interlock Pipeline Stages) using VHDL. It also describes the instruction set, architecture and timing diagram of the processor. Floating point number to fixed number conversion is the main task while working on this numbers, this conversion has been achieved by using Float to Fixed number converter module. Finally, design, synthesis and simulation of the proposed RISC Processor based on MIPS has been achieved using Xilinx ISE 13.1i Simulator and coding is written in VHDL language.**

Keywords—Architecture; Instruction Set; RISC; VHDL; XILINX 13.1i.

I. INTRODUCTION

Nowadays, designing of RISC processors is been a trend that are efficient for various application in terms of speed, area, etc. Performance is the main criteria for designing of any processor. The proposed RISC processor based on MIPS designed here is an effort towards efficient processor suitable for various applications.

CISC processors have received the marketplace over the years. They support various addressing modes and various data types like others complex processors. Length of instruction varies from instruction to instruction. They generally access data from external memory. They are basically implemented using micro programmed control. There is little gap between instructions of CISC processor and higher-level language statements. However, they may save memory space, its design is complicated and instructions are variable in length; special hardware is required for boundary marking of instruction. After an deep study it is proved that simple instructions has been used 80% of the time and complex instructions has been replaced by group of simple instructions.

On the other hand, RISC processor requires very few data types and performs the simple operations. It also supports very few addressing modes and mostly based on registers. Many of the instructions operate on data which are present in internal registers. LOAD and STORE instructions are the only instructions which access data from external memory. Here decoding becomes easier, since the instruction length is fixed.

Execution of instructions in parallel using the pipelined stages will improves the overall throughput of the processor but it will introduces some of the hazards in its working operation. Data hazards are those hazards which are generated due to sharing of source and destination resources in succeeding instructions, this will happen in the case when the source for an instruction is destination for previous instruction. This can be prevented by using the forwarding method. Structural hazards are generated when the program and data memory is used commonly. By designing the prefetch queue in processor, structural hazards can be removed. Control hazards are generated in non-sequential executed circuits and to remove this hazards flushing method is used [5].

RISC architecture's analysis gives many important issues in computer architecture. Most RISC computers have the same key elements:

- A limited and simple instruction set.
- A large number of GPR's (General Purpose Registers).
- Optimization of the instruction pipeline.

Organization of RISC computers can be represented by three main components:

- ALU (Arithmetic Logic Unit): performs the actual computation and processing of data.
- CU (Control Unit): controls the movement of data and instructions into and out of the CPU and controls the operation of the ALU.
- RS (Register Set): a minimal internal memory, which consists of a set of storage locations [6].

Comparing to CISC, RISC CPU have more advantages, such as faster speed, simplified structure easier implementation. RISC CPU is extensive use in embedded system. Therefore, designing of RISC CPU based on MIPS is the necessary choice.

[978-1-4799-6438-3/15/$31.00 ©2015 IEEE]

II. INSTRUCTION SET ARCHITECTURE (ISA)

The instructions of RISC Processor based on MIPS are categories into four different instruction formats; R-Type, I-Type, J-Type and I/O Type as described in figures below;

a) R-Type Instruction Format

(31 to 26)	(25 to 21)	(20 to 16)	(15 to 11)	(10 to 6)	(5 to 0)
Opcode	Rs	Rt	Rd	Shamt	Function

Figure 1. R-Type Instruction Format

Figure 1 shows Register type instruction format. This type of format has total six fields; Opcode field of 6-bit, used to select the type of instruction format, Rs (Source register), Rt (Target register) and Rd (Destination register) are of 5-bit each which are used for storage of data. Shamt (Shift amount) field of 5-bit, used for data shifting and last is the Function field of 6-bit used for selection of different functions to be performed.

b) I-Type Instruction Format

(31 to 26)	(25 to 21)	(20 to 16)	(15 to 0)
Opcode	Rs	Rt	Address/Immediate Value

Figure 2. I-Type Instruction Format

Figure 2 shows Immediate type instruction format. This type of format has total four fields, Opcode field of 6-bit, used to select the type of Instruction format, Rs (Source register) and Rt (Target register) are of 5-bit each, used for storage of data and the last is Address/Immediate Value field of 16-bit, used for immediate data operations.

c) J-Type Instruction Format

(31 to 26)	(25 to 0)
Opcode	Target Address

Figure 3. J-Type Instruction Format

Figure 3 shows J-type instruction format. This type of format has only two fields; Opcode field of 6-bit, used to select the type of Instruction format. Target address of 26-bit, used to specify on which address to jump.

d) I/O-Type Instruction Format

(31 to 26)	(25 to 21)	(20 to 16)	(15 to 0)
Opcode	Rs	Rd	Immediate Value

Figure 4. I/O-Type Instruction Format

Figure 4 shows I/O-type instruction format. This type of format has total four fields; all the fields are same as in R-Type and I-Type instruction formats. Instructions which require read and write of data from port uses this type of instruction format. Execution of any instruction becomes faster by using such types of instruction formats.

Since, the designed RISC Processor based on MIPS is of 32-bit. Hence, all the instructions designed are 32-bit in length and use the different instruction formats for execution of instructions. It also supports addressing modes viz. Implicit addressing, Immediate addressing mode, Register addressing mode, direct addressing mode, and Register indirect addressing.

III. RISC ARCHITECTURE BASED ON MIPS

Figure 5 shows the internal architecture of 32-bit RISC processor based on MIPS. It consists of memory unit, decoder unit, execution unit, register unit, address/data bus and pipeline stages. Register unit consists of MIPS registers, flag Register, program counter and stack pointer. Similarly, execution unit consists of control unit/operation select, floating point conversion unit, floating point ALU and result to destination unit.

The various instructions are executed in the following manner; opcode is fetch from the memory and given to pipeline stages, then after pipelining it is given to decoder unit, then execution unit and register unit. Finally, the result will be store by result to destination unit.

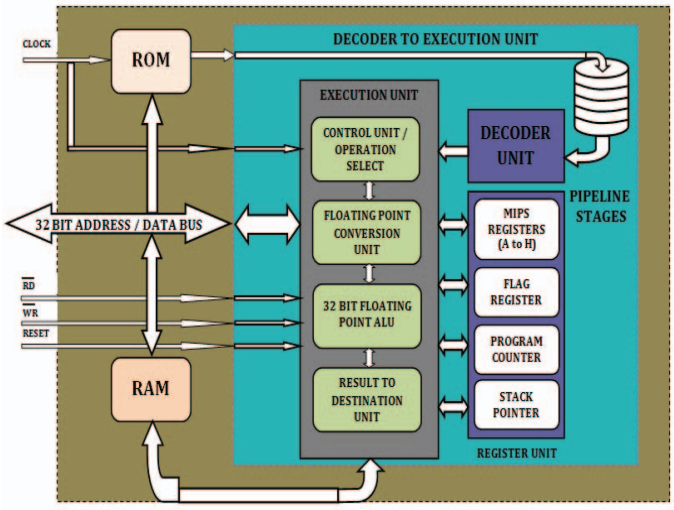

Figure 5. Internal Architecture of MIPS RISC Processor

a) ROM

Read Only Memory of RISC processor based on MIPS is of 32-bit. ROM contains the opcode of the instructions to be executed. The opcode of the instruction is fetch and given to the pipeline stages.

b) Pipeline Stages

It contains five pipeline stages; IF (Instruction Fetch), ID (Instruction Decode), OF (Operand Fetch), IE (Instruction Execute) and WB (Write Back).

c) Decoder Unit

After the pipeline stages, the output is given to decoder unit. Decoder unit has different instruction formats, depending on the type of opcode instruction formats will be used and instruction will be decoded.

d) Register Unit

The register unit of this system contains following registers;

(i) MIPS Registers

MIPS architecture defines eight 32-bit general purpose registers which are also called as MIPS Registers viz. Reg A, Reg B, Reg C, Reg D, Reg E, Reg F, Reg G, and Reg H, which are used for load and store operations.

(ii) Flag Register

RISC Processor based on MIPS consist of 32-bit flag register. It has 32 flip flops, out of which 6 flip flops are defined as flag and rest of the 28 flip flops are undefined. Hence kept as don't care. Figure 6 shows the structure of flag register.

(31 to 6)	(5)	(4)	(3)	(2)	(1)	(0)
X	CF	ACF	SF	OVF	ZF	PF

Figure 6. Structure of Flag Register

(1) PF (Parity Flag)

When MIPS performs 16-bit or 32-bit Floating Point arithmetic or logical operation and the result obtained in Floating Point ALU contains even number of 1's then it is called as even parity result, and parity flag becomes one (PF = 1). But, if the number of 1's obtain in result are odd then it is called as odd parity result, and parity flag remains zero (PF = 0).

(2) ZF (Zero Flag)

When MIPS performs 16-bit or 32-bit Floating Point arithmetic or logical operation and if the result obtained in Floating Point ALU is zero (0000H or 00000000H) then zero flag will become one (ZF = 1). But if the result obtained in Floating Point ALU is non-zero (0001H to FFFFH or 00000001H to FFFFFFFFH) then zero flag remains zero indicating result is non-zero (ZF = 0).

(3) OVF (Overflow Flag)

When MIPS performs 16-bit or 32-bit Floating Point arithmetic or logical operation and if there is carry into MSB as well as carry out of MSB then overflow flag remains zero (OVF = 0), indicating result is not overflowed. But if there is carry into MSB but no carry out of MSB or if there is no carry into MSB but carry out of MSB then overflow flag becomes one (OVF = 1), indicating result is overflowed.

(4) SF (Sign Flag)

When MIPS performs 16-bit or 32-bit Floating Point arithmetic or logical operation on signed number and if the result obtained is out of the range of signed number then it is called as incorrect result and signed flag is indicated the incorrect result i.e. if MSB of the result is one then signed flag remains one indicating result is negative (SF = 1). But if the result obtained is within the range of signed number then it is called as correct result and signed flag is indicated the correct result i.e. if MSB of the result is zero then signed flag remains zero indicating result is positive (SF = 0). For unsigned number operations sign flag is of no use.

(5) ACF (Auxiliary Carry Flag)

When MIPS performs 16-bit or 32-bit Floating Point arithmetic operation and if there is a carry generated from LSB or borrow required for LSB then auxiliary carry flag becomes one (ACF = 1). But if there is no carry generated from LSB or no borrow required for LSB then auxiliary carry flag remains zero (ACF = 0).

(6) CF (Carry Flag)

When MIPS performs 16-bit or 32-bit Floating Point arithmetic operation and if there is a carry generated from MSB or borrow required for MSB then carry flag becomes one (CF = 1). But if there is no carry generated from MSB or no borrow required for MSB then carry flag remains zero (CF = 0).

(iii) Program Counter

It is a 32-bit register used to store 32-bit address of that memory location from which an opcode is to be fetched or readed. After every opcode fetched or readed it is auto incremented by 1 or 2 (depending on instruction) so as to read the opcode of next instruction.

(iv) Stack Pointer

Placing of anything one above the other is nothing but stack. Stack pointer is a 32-bit register used to store 32-bit address of stack top memory location. After each PUSH operation it is auto decremented by 2, then stack pointer will store the 32-bit address of new stack top memory location.

e) Execution Unit

The execution unit of this system contains the following sub-units;

(i) Control Unit / Operation Select

Decoder unit gets the input from pipeline stages and give it to execution unit. Control unit gives control signals to those block which are involve in the current operation. It also selects the operation to be performed.

(ii) Floating Point Conversion Unit

This unit converts the number into single precision IEEE 754 floating point format for further calculations.

(iii) 32-bit Floating Point ALU

This unit performs arithmetic as well as logical operations on maximum 32-bit at a time. Arithmetic operations performed by IEEE 754 standards are as follows;

(1) Floating Point Addition / Subtraction Algorithm

Assuming that the operands are already in the IEEE 754 format, performing floating point addition / subtraction:

Result = X + Y = (Xm x 2Xe) + (Ym x 2Ye) or

Result = X - Y = (Xm x 2Xe) - (Ym x 2Ye)

The following steps are carried out by this algorithm:

1) Align Binary Point

- Initial result exponent: the larger of Xe, Ye.
- Compute exponent difference: Ye – Xe.
- If Ye > Xe Right shift Xm that many positions to form Xm 2 Xe - Ye.
- If Xe > Ye Right shift Ym that many positions to form Ym 2 Ye - Xe.

2) Compute Sum of Aligned Mantissas

- Xm2 Xe-Ye + Ym or Xm + Xm2 Ye-Xe.

3) If Normalization of Result is needed, then Normalization Step Follows

- Left shift result, decrement result exponent (e.g., if result is 0.001xx…) or
- Right shift result, increment result exponent (e.g., if result is 10.1xx…) Continue until MSB of data is 1.

4) Check Result Exponent

- If larger than maximum exponent allowed return exponent overflow.
- If smaller than minimum exponent allowed return exponent underflow.

5) If result mantissa is 0, may need to set the exponent to zero by a special step to return a proper zero.

The flow chart for floating point addition / subtraction is shown in figure below.

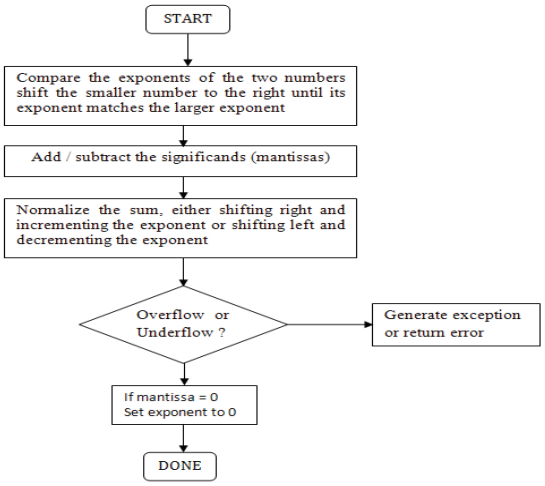

Figure 7. Flow Chart for Floating Point Addition / Subtraction

(2) Floating Point Multiplication Algorithm

Assuming that the operands are already in the IEEE 754 format, performing floating point multiplication:

Result = R=X * Y = (-1)Xs (Xm x 2Xe) * (-1)Ys (Ym x 2Ye)

The flow chart for floating point multiplication is shown in figure below.

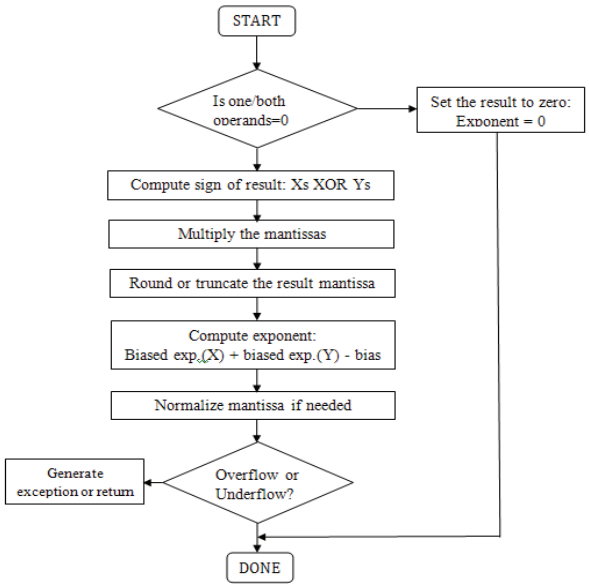

Figure 8. Flow Chart for Floating Point Multiplication

This algorithm involves the following steps:

1) If one or both operands is equal to zero, return the result as zero, otherwise.

2) Compute the sign of the result Xs XOR Ys.

3) Compute the mantissa of the result:

- Multiply the mantissas: Xm * Ym.
- Round the result to the allowed number of mantissa bits.

4) Compute the exponent of the result:

Result exponent = biased exponent (X) + biased exponent (Y) – bias.

5) Normalize if needed, by shifting mantissa right, incrementing result exponent.

6) Check result exponent for overflow/underflow:

- If larger than maximum exponent allowed return exponent overflow.
- If smaller than minimum exponent allowed return exponent underflow.

(3) Floating Point Division Algorithm

Assuming that the operands are already in the IEEE 754 format, performing floating point division:

Result = R = X / Y = (-1)Xs (Xm x 2Xe) / (-1)Ys (Ym x 2Ye)

This algorithm involves the following steps:

1) If one or both operands is equal to zero, return the result as zero, otherwise.

2) Compute the sign of the result Xs XOR Ys.

3) Compute the mantissa of the result:

- Multiply the mantissas: Xm / Ym.
- Round the result to the allowed number of mantissa bits.

4) Compute the exponent of the result:

Result exponent = biased exponent (X) + biased exponent (Y) + bias.

5) Normalize if needed, by shifting mantissa right, incrementing result exponent.

6) Check result exponent for overflow/underflow:

- If larger than maximum exponent allowed return exponent overflow.
- If smaller than minimum exponent allowed return exponent underflow.

The flow chart for floating point division is shown in figure below.

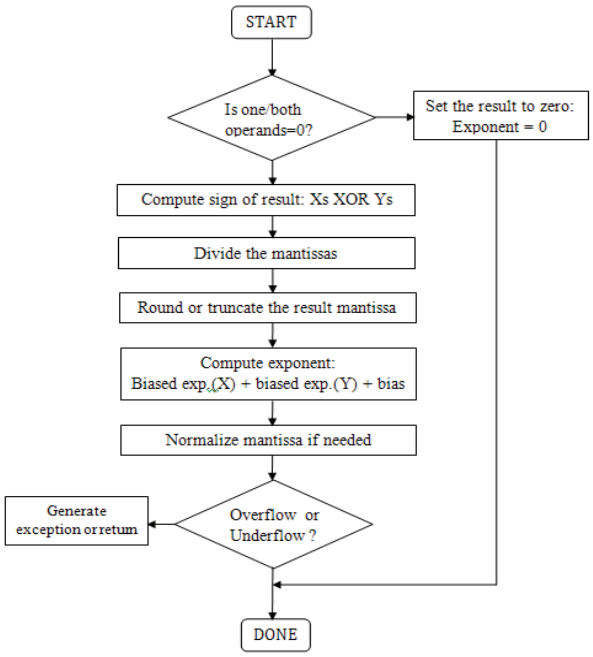

Figure 9. Flow Chart for Floating Point Division

(iv) Result to Destination Unit

After execution of any instruction, the result must be store into destination. This mechanism is done by this unit. The destination can be various MIPS registers, memory location of 32-bit address, etc.

f) RAM

Random Access Memory of RISC processor based on MIPS is of 32-bit. It stores the data while PUSH operation and other stack related operation.

IV. TIMING DIAGRAM

The timing diagram of RISC processor based on MIPS shows the timing of different pipeline stages viz. instruction fetch, instruction decode, operand fetch, instruction execute and result to destination as shown in figure 10. It also shows the pipeline design of RISC processor based on MIPS.

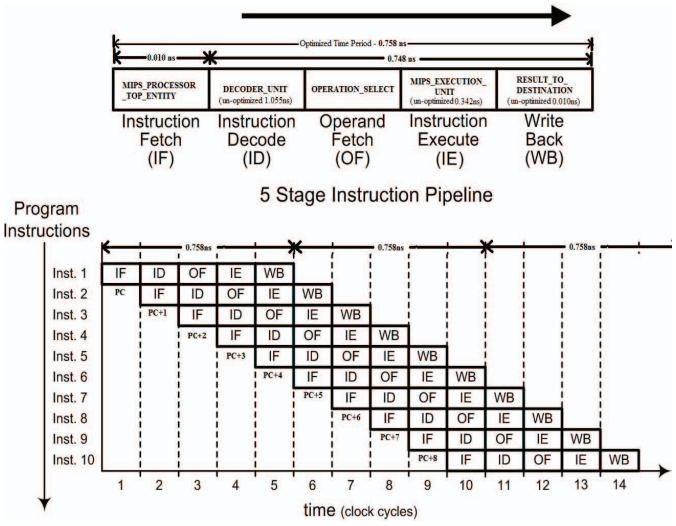

Figure 10. Timing Diagram of MIPS RISC Processor

From figure it has been shown that IF requires delay of 0.01ns. ID, OF, IE and WB stage i.e. Decoder to execution unit requires delay of 0.748ns. Therefore, total can be calculated as 0.758ns.

V. EXPERIMENTAL RESULTS

A. RTL View

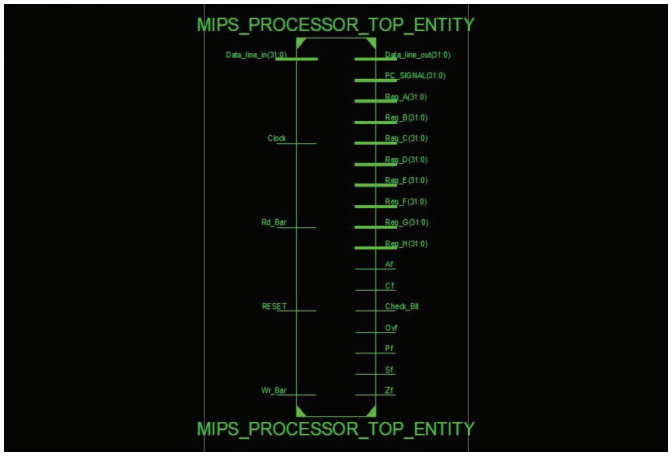

Figure 11. RTL View of MIPS RISC Processor

Figure 11 shows the RTL (Register Transfer Logic) view of 32-bit RISC Processor based on MIPS. It has input pins; Data_line_in, Clock, Rd_Bar, RESET, Wr_Bar and output pins; Data_line_out, PC_SIGNAL, Reg_A, Reg_B, Reg_C, Reg_D, Reg_E, Reg_F, Reg_G, Reg_H, Af, Cf, Check_Bit, Ovf, Pf, Sf and Zf.

It shows an instruction fetch unit, instruction decoder unit, execution unit and memory unit. Instruction fetch unit is used to fetch or read opcode from memory using PC (program Counter) and then pass the result to instruction decoder unit. Instruction decoder unit is used to receive the opcode comes from instruction fetch unit. Depending on the type of opcode, instruction format will be selected and then it will be given to execution unit for performing operation of the instruction. Execution unit is used to perform the various operations depending upon the opcode of the instruction. This unit comprises of ALU based on floating point arithmetic, Flag register of 32-bit, instruction set, MIPS registers for data storage and Control unit (operation select). Memory unit is used to store the opcodes and for stack related operations.

B. Simulation Result

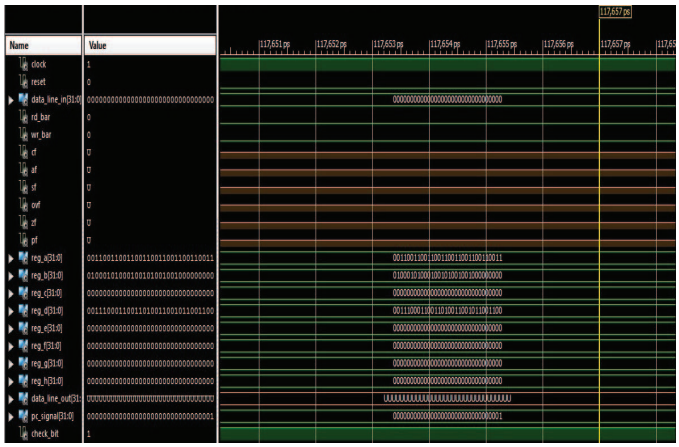

Figure 12. Simulation Result of MIPS RISC Processor

Figure 12 shows the simulation result of RISC Processor based on MIPS. Instruction MUL D, A, B has been performed here. The operation is carried out such that, firstly register A data ("00110011001100110011001100110011") and register B ("01000101000100101001001000000000") data is readed and then floating point multiplication of this two registers data is done and then the result is stored into the destination register D ("00111000110011010011001011001100").

VI. CONCLUSION AND FUTURE SCOPE

In this paper, we use top-down design method in which we design instruction fetch unit, decoder unit, operand fetch unit, execution unit and memory unit. VHDL language has been used to describe the system. Pipelined design has been achieved with less clock cycles per instruction. Through pipelining, the maximum throughput of execution is achieved as one instruction per machine cycle. This is possible due to hardwired approach for design of control unit and fixed length instruction format. To resolve data hazards, result forwarding is efficient than stalling as it remove the penalty of time in handling such conflicts. The 32-bit RISC processor based on MIPS with an instruction set has been designed. Every instruction is executed based on pipeline approach i.e. Instruction Fetch, Decode, Operand Fetch, Execution and then Write Back. The design is verified through exhaustive simulations. Some of the applications are presented. The design, synthesis and simulation of the RISC processor based on MIPS has been achieved using Xilinx 13.1i ISE Simulator. The 32-bit RISC Processor based on MIPS has achieved combinational delay of 0.758ns and maximum operating frequency of 1.350 GHz.

If we suppose to implement this processor using FPGA technology it is possible to upgrade the system with new features as per user requirements. The hardware complexity can be reduced and integration of different circuits in a single chip can be possible on FPGA kit. In the near future applications like the ATMs, mobile phones, portable gaming kits can be implemented. Although the future belongs to the super pipelined/superscalar processors, CISC designs will not disappear so fast, just because of the enormous installed base of such computers. RISC and CISC will peacefully coexist until CISC adopts so many features of RISC that it will be hard to tell the difference between these two processors. So, in future developing a RISC Processor is necessary choice because of its reduced instruction set.

REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] Pravin S. Mane, Indra Gupta, M. K. Vasantha, Implementation of RISC Processor on FPGA, 1-4244-0726-5/06, 2006 IEEE.

[6] Jarrod D. Luker and Vinod B. F'rasad, RISC System Design in an FPGA, 0-7803-7150-X/01/$10.00@20011 IEEE.

[7] Shuchita Pare, Dr. Rita Jain, 32Bit Floating Point Arithmetic Logic Unit ALU Design and Simulation, Dec 2012, IJETECS.

[8] Bai-ZhongYing, Computer Organization, Science Press, 2000.11.

[9] Wang-AiYing, Organization and Structure of Computer, Tsinghua University Press, 2006.

[10] Wang-Yuan Zhen, IBM-PC Macro Asm Program, Huazhong University of Science and Technology Press, 1996.9.

[11] MIPS Technologies, Inc. MIPS32™ Architecture For Programmers Volume II: The MIPS32™ Instruction Set June 9, 2003.

[12] Zheng-WeiMin, Tang-ZhiZhong, Computer System Structure (The second edition), Tsinghua University Press, 2006.

[13] Mr. Sagar P. Ritpurkar, Prof. Mangesh N. Thakare, Prof. Girish D. Korde, Review on 32Bit MIPS RISC Processor using VHDL, International Conference on Advances in Engineering & Technology – 2014 (ICAET-2014), PP 46-50, IOSR.