



Design and Implementation of RISC Processor on FPGA

¹Vishwas V. Balpande*, ²Vijendra P. Meshramm, ³Ishan A. Patilm, ⁴Sukeshini N. Tamgadem, ⁵Prashant Wanjari

^{1, 2, 3, 4}Electronics Engineering, DBACER, Nagpur, Maharashtra, India

⁵Electronics Engineering, RG CER, Nagpur, Maharashtra, India

Abstract— A true 16-bit RISC processor has been designed using VHDL. Hierarchical approach has been used so that basic units can be modeled using behavioral programming. These basic units are combined using structural programming. Four stage (viz. instruction fetch stage, instruction decode stage, execution stage and memory/IO - write back stage) pipelining is used to improve the overall CPI (Clock Cycles per Instruction). Hardwired control approach is used to design the control unit as against microprogrammed control approach in conventional CISC processor. The processor has one input port, one output port and six hardware vectored interrupts along with 16-bit address bus and 16-bit data bus. Structural hazards are dealt with the implementation of prefetch unit, data hazards are dealt with forwarding and control hazards are dealt with flushing and stalling. The design has been implemented on FPGA for verification purpose.

Keywords— VHDL, CPI, RISC, CISC, FPGA

I. INTRODUCTION

A trend is towards the designing of RISC processors that are efficient for specific application and meet the minimum requirements of application in hand [1]. Performance is the main criteria for designing of such processors [2]. The proposed RISC processor designed here is an effort towards efficient processor suitable for small applications.

CISC processors have gained the common marketplace over the years. They support various addressing modes and various data types. The instruction length varies from instruction to instruction. They frequently access data in external memory. They are generally implemented using microprogrammed control. There is little semantic gap between instructions of CISC processor and statements in higher-level languages. Although they may be saving memory space, designs is complicated and as instructions are variable in length, special hardware for boundary marking of instruction is required. Study over the years proved that simple instructions are used 80% of the time and many complex instructions can be replaced by group of simple instructions [3].

RISC processor operates on very few data types and does the simple operations. It supports very few addressing modes and are mostly register based. Most of the instructions operate on data present in internal registers. Only LOAD and STORE instructions access data in external memory. Also the instruction length is fixed and hence decoding is easier.

Parallel execution o instructions through the pipelined stages of processor improves the overall throughput of the processor but introduces some hazards in its working [4]. Data hazards are due to sharing of destination and source resources in succeeding instructions (source for an instruction is destination for previous instruction) and they can be resolved by the method of forwarding. Structural hazards are due to common program and data memory. By implementing the prefetch queue in processor, structural hazard can be handled. Control hazards are introduced in non-sequential execution of an instruction and the method of flushing is used to deal with them.

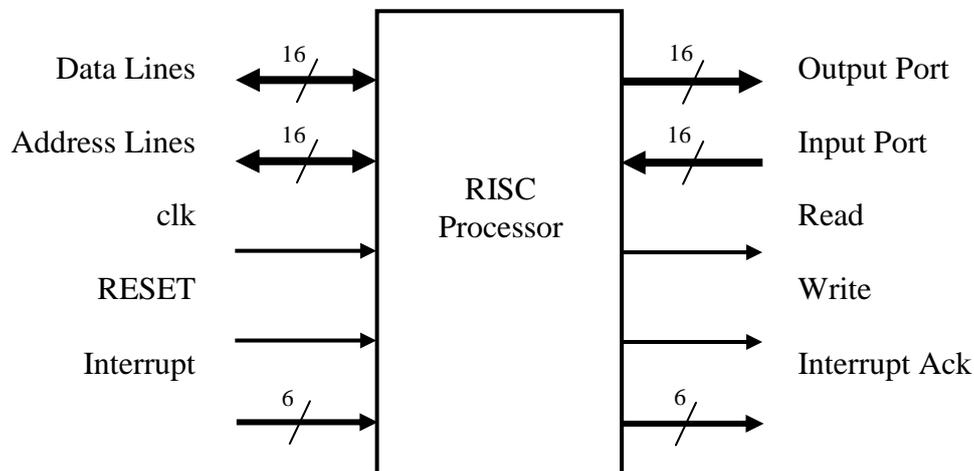


Fig. 1. RISC Processor

Proposed RISC processor has four stages in the pipeline and in general each stage take one clock pulse to complete its operation. It has 16-address lines, 16-data lines, one 16-bit input port and one 16-bit output port. The logical block of it is shown in fig. 1. All internal registers are 16-bit in length.

II. INSTRUCTION SET ARCHITECTURE(ISA)

As a true 16-bit RISC processor, all instructions are 16-bit in length and use the 3-operand notation method (2-source operands and 1-destination operand). The design has 8-internal general purpose registers all are 16-bit in length. The general instruction format for the designed RISC processor is given in fig. 2. Proposed processor supports 37-instructions as summarized in table. It supports addressing modes viz. register addressing, immediate addressing and registers indirect addressing.

The actual instruction format may vary from instruction to instruction. As each instruction has different format, they follow different datapath, which are combined together to form final datapath.



Fig. 2. General Instruction Format

Table 1 Instruction Set Summary

Instruction	Description	Instruction	Description
ADD RD, RS1, RS2	Signed addition, $RD = RS1 + RS2$	OUT RS	Output port = data in register RS
ADDu RD, RS1, RS2	Unsigned addition, $RD = RS1 + RS2$	BZ RD, RS	Branch on zero, If $RS = 0$ jump to loc RD
SUB RD, RS1, RS2	Signed subtraction, $RD = RS1 - RS2$	BNZ RD, RS	Branch not zero, If $RS \neq 0$ jump to loc RD
SUBu RD, RS1, RS2	Unsigned subtraction, $RD = RS1 - RS2$	EI D6	Enable interrupt whose value is 1 in D6
ADDI RD, RS1, D6	Signed addition, $RD = RS1 + D6$	JAL RD, RS	Jump to loc RS and link back to loc RD
SUBI RD, RS1, D6	Signed subtraction, $RD = RS1 - D6$	RJAL RD	Link back to loc RD
SLT RD, RS1, RS2	Signed set less than, $RD = 0$ if $RS1 < RS2$	RETI	Return from interrupt to loc in TRAP register
SLTu RD, RS1, RS2	Unsigned set less than, $RD = 0$ if $RS1 < RS2$	MVIL RD, D8	Move D8 in lower byte of RD
NOT RD, RS	Logical NOT, $RD = \text{NOT}(RS)$	MVIH RD, D8	Move D8 in upper byte of RD
AND RD, RS1, RS2	Logical AND, $RD = RS1 \text{ AND } RS2$	BZI RD, D8	If $RD = 0$, jump to loc = $(PC) + D8$
OR RD, RS 1, RS2	Logical OR, $RD = RS1 \text{ OR } RS2$	BNZI RD, D8	If $RD \neq 0$, jump to loc = $(PC) + D8$
XOR RD, RS 1, RS2	Logical XOR, $RD = RS1 \text{ XOR } RS2$	SLLI RD, RS, D5	$RD = RS$ shifted left logically by D5
NOR RD, RS 1, RS2	Logical NOR, $RD = \text{NOT}(RS1 \text{ OR } RS2)$	SRLI RD, RS, D5	$RD = RS$ shifted right logically by D5
SLL RD, RS1, RS2	Logic shift left, $RD = RS1$ shifted by $RS2$	SRAI RD, RS, D5	$RD = RS$ shifted right arithmetic by D5
SRL RD, RS1, RS2	Logic shift right, $RD = RS1$ shifted by $RS2$	RORI RD, RS, D5	$RD = RS$ rotated right by D5
SRA RD, RS1, RS2	Arithmetic shift right, $RD = RS1$ shifted by $RS2$	LW RD, RS, D6	$RD = \text{data from memory loc } (RS + D6)$
ROR RD, RS1, RS2	Rotated right, $RD = RS$ rotated by $RS2$	SW RD, RS, D6	Loc $(RS + D6)$ in memory = RD
IN RD	$RD = \text{data on input port}$	HLT	Halt the program execution

III. PROCESSOR ARCHITECTURE

The proposed architecture consists of pipelined unit, control unit, hazard detection unit, branch forwarding unit, execution forwarding unit, interrupt and exception unit and prefetch unit. Its block diagram is shown in fig.3. Pipelined unit consists of four stages viz. instruction fetch unit, instruction decode unit, instruction execution unit and memory/IO-write back unit.

1. *Pipelined unit:* The pipelined unit designed for this processor consists of four stages viz. instruction fetches stage, instruction decodes stage, instruction execution stage and memory/IO-write back stage. Each stage is connected to the next stage through output register of each stage to implement just-in-time principle. Instruction fetch stage fetches instructions from memory through prefetch unit and consists of program counter and selector to choose correct PC value based on current instruction in execution stage. For sequential execution of program, it selects incremented program counter value. For non-sequential execution, it selects either branch target address or vector address for interrupts and exceptions. It also has the program counter incrementer. Instructions decode stage separates the opcode, function and the source and destination codes for control unit. Register file is a part of this unit, which consists of 8-general purpose registers. If the instruction has immediate data, it is signed extended to 16-bit by this stage. This stage also responsible for differentiating between absolute branching and relative branching. Instruction execution stage does all the arithmetic and logical operations including shift and rotate. The arithmetical (addition, subtraction etc.) and logical (logical AND, OR, XOR etc.) operations are carried out by basic ALU, shift and rotate operations are carried out by shift unit and move immediate unit load immediate data in specified register. Arithmetic logic unit also sets the overflow flag in case of arithmetic instructions. Memory/IO-write back stage is responsible for accessing external memory for data and for writing the results back into destination register. In case of IN instruction, data on input port is directly sent to destination register by this unit. It also place on output port for OUT instruction.

2. *Control Unit:* Hardwired control approach is best suited for control unit of RISC processor because very few control signals needs to be generated for smaller set of instructions. Sequencing of control signals for various stages of processor is important issue and is achieved by passing them through clock sensitive units in succession. The block diagram of hardwired control unit designed is shown in figure 4.

3. *Hazard Detection Unit:* This unit primarily detects the resource conflicts and generates the necessary signals for execution forwarding unit and branch forwarding unit. The source of current instruction may be the destination of previous instruction and in such case the data supplied to execution stage is not correct until result of previous instruction is written back to destination.

4. *Branch Forwarding Unit:* The branch target address for any branch instruction or conditional data for conditional branch instruction may be the destination for previous instruction. When an interrupt occurs and the current instruction in execution is branch instruction then the return address is not the next sequential address but it will be the branch target address. This unit takes care of all such situations. It also flushes the instruction following the branch instruction in the pipeline if branching actually takes place. If data hazards occur for branch instruction then result from execution unit is selected as a component for branch target address calculation for any branch instruction or conditional data for conditional branch instruction.

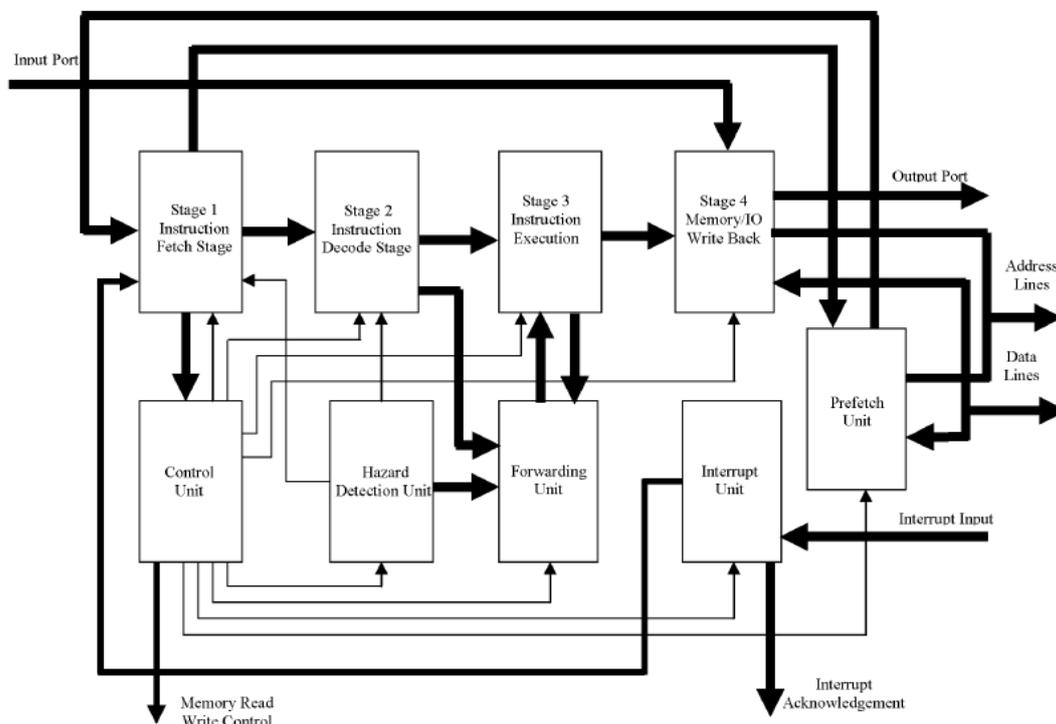


Fig.3. RISC Processor Architecture Block Diagram

5. *Execution Forwarding Unit:* When data hazards occurs this unit directly forwards the result of execution unit or results from write-back stage as a source to the instruction following the current instruction in execution stage.

6. *Interrupt and Exception Unit:* RESET has the highest priority, and then overflow exception, hardware interrupt lines 5 to 0 has priority in decreasing order. All of them are vectored. Although zero flag, carry flag etc. are not designed, they can be simulated through exception routine for overflow exception and branch instructions. The return address for exceptions and hardware interrupts is stored in TRAP register, which is part of this unit.

7. *Prefetch Unit*: This unit makes use of idle bus time to prefetch instructions from memory and stores them in prefetch queue. Prefetch queue has the capacity of storing four instructions at a time along with tag for each of them. The tag specifies from which memory location the corresponding instruction has been fetched. The prefetch unit works like small look-ahead cache. It does not work on FIFO principle. This unit supplies instruction to instruction fetch stage.

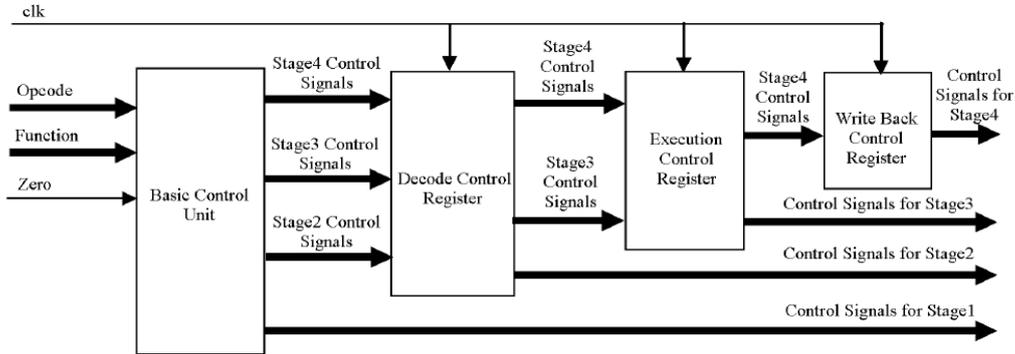


Fig.4. Block Diagram of Hardwired Control Unit

IV. VHDL

VHDL is acronym for Very High Speed Integrated Circuit Hardware Description Language. It is designed to fill a number of needs in the design process. First it allows description of the structure of system that is, how it is decomposed into subsystems and how they are interconnected. Second, it allows the specification of the function of a system using familiar programming language forms. Third, as a result, it allows the design of a system to be simulated before being manufactured, so that designers can quickly compare alternatives and test for correctness without the delay and expense of hardware prototyping. Fourth, it allows the detailed structure of a design to be synthesized from a more abstract specification, allowing designers to concentrate on more strategic design decisions and reducing time to market [6]. VHDL was established as the IEEE 1076 standard in 1987. In 1993, the IEEE 1076 standard was updated and an additional standard, IEEE 1164 was adopted. In 1996, IEEE 1076.3 became the VHDL synthesis standard.

V. IMPLEMENTATION ON FPGA

FPGA (Field Programmable Gate Arrays) is a device used for the verification of design. It works as raw IC where user can implement its design on it and verify the correctness of design. This allows for cheaper prototyping and shorter time to-market of hardware designs. It consists of LUTs (Look Up Tables), CLBs (Configurable Logic Blocks), LCs (Logic Cells) and JOBs (hinput Output Blocks) [7]. The designed processor has been implemented on Xilinx's Spartan-II FPGA for verification purpose. The device utilization summary is given table II.

VI. SIMULATION RESULTS

After modeling the design through VHDL, it is simulated before implementing it on FPGA. Xilinx's ISE simulator is used for this purpose. The input test bench waveform is used as source for applying the input values and the output waveform from simulator is used as a mean to verify the correctness of the design [5]. Simulation is used as debugging tool for correcting errors in design. Sample waveform for execution of small program is given in fig 5. This program is for addition of hexadecimal data 1010H and 3030H loaded in register R1 and R2 and the addition is placed in R0 register. The resource conflict has occurred in this program because destination of MOV instruction is R1 and source for next succeeding instruction ADD is again R1. This resource conflict is successfully handled as shown in simulation results.

Table 2 : Device utilization summary

Logic Utilization	Used	Availabl e	Utilization
Total Number Slice Registers	570	4,704	12%
Number used as Flip Flop	48		
Number used as Latches	522		
Number of 4 input LUTs	2,001	4,704	42%
Logic Distribution			
Number of occupied Slices	1,137	2,352	48%
Number of Slices containing only related logic	1,137	1,137	100%
Number of Slices containing only unrelated logic	0	1,137	0%
Total Number of 4 input LUTs	2,123	4,704	45%
Number used as logic	2,001		

Number used as route-through	122		
Number of bonded IOBs	91	140	65%
Number of GCLKs	1	4	25%
Number of GCLKIOBs	1	4	25%

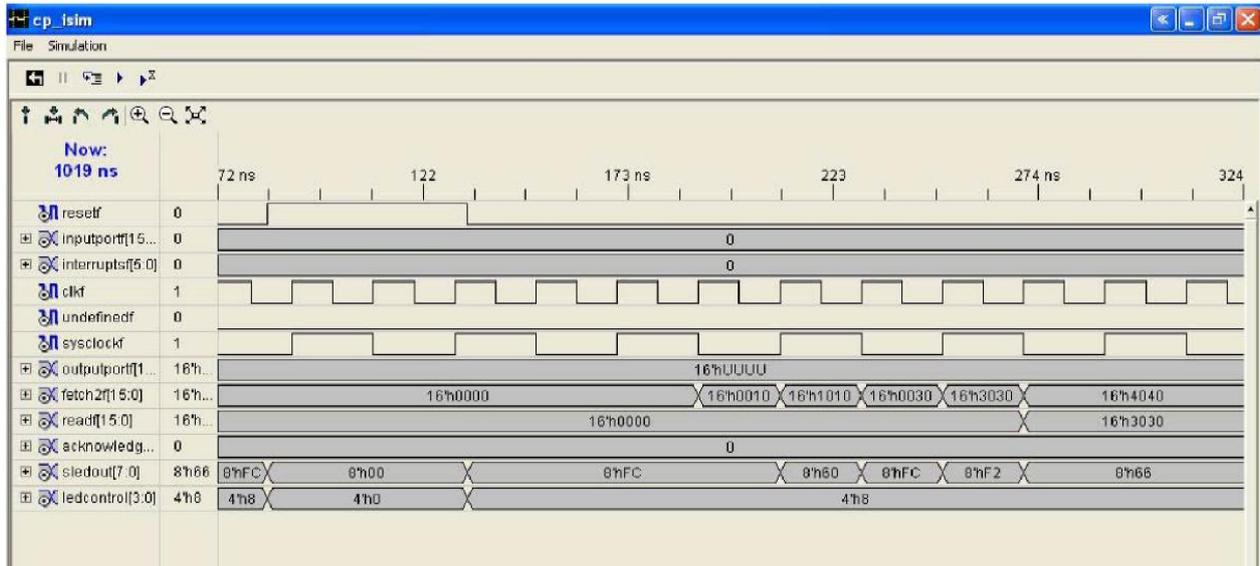


Fig. 5. Simulation Results of Designed RSIC Processor Connected to Memory

VII. CONCLUSION

Through pipelining, the maximum throughput of execution is achieved as one instruction per clock pulse provided that there are no stalls. 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 prefetch unit is used for handling the structural hazards while flushing is used to handle control hazards. The prefetch buffer is like a small cache storing tag along with instruction fetched. It does not work on FIFO principle. The design is modeled and simulated using VHDL and then implemented on FPGA successfully. The maximum frequency of operation on the Xilinx's Spartan-II FPGA is 26-MHz.

REFERENCES

- [1] M. A. S.D. Poz, J. E. A. Cobo, W. A. M. V. Noije and M. K. Zuffo, "A Simple RISC Microprocessor Core Designed for Digital Set-Top-Box Applications," IEEE International Conference on Application-Specific Systems, Architectures, and Processors, 2000. Proceedings, pp. 35-44.
- [2] Xia Li, Longwei Ji, Bo Shen, Wenhong Li and Qianling Zhang, "VLSI Implementation of a High-performance 32-bit RISC Microprocessor," International Conference on Communications, Circuits and Systems and West Sino Expositions, IEEE 2002, Vol. 2, pp.1458-1461
- [3] Sivarama P. Dandamudi, "Fundamentals of Computer Organization and design" Springer-Verlag New York, Inc., 2003.
- [4] John L. Hennessy and David A. Patterson, "Computer Architecture A Quantitative Approach," 3rd Edition, Morgan Kaufmann Publishers, 2003.
- [5] Zhenyu Gu, Zhiyi Yu, Bo Shen and Qianling Zhang, "Functional Verification Methodology of a 32-bit RISC Processor," International Conference on Communications, Circuits and Systems and West Sino Expositions, IEEE 2002, Vlo. 2, pp. 1454-1457
- [6] Peter J. Ashenden, "The Designer's Guide to VHDL," 2nd Edition, Morgan Kaufmann Publishers, 2002.
- [7] Spartan-II Platform FPGA handbook October 24, 2002.