International Journal of Scientific Research in Physics and Applied Sciences
Research Paper Vol-1, Issue-1 ISSN: XXXX-XXXX

Design of 16-bit RISC Processor

Supraj Gaonkar[1], Anitha M.[2*]

[1]*Sir M Visvesvaraya Institute of Technology Bangalore, Karnataka, India*
[*2] *Department of Tele. Engg., Sir M Visvesvaraya Institute of Technology Bangalore, Karnataka, India*
e-mail:supraj@outlook.com,anithapadela123@gmail.com

Abstract— The Reduced Instruction Set Computer or RISC is a microprocessor design principle that favours a smaller and simpler set of instructions that all take same amount of time to execute. RISC architecture is used across a wide range of platforms from cellular phones to super-computers. In this paper the behavioural design and functional characteristics of 16-bit RISC processor is proposed, which utilizes minimum functional units without compromising in performance. The design is based on Harvard architecture having separate data memory and instruction memory. The instruction word length is 24-bit wide. The processor supports 16 instructions with three addressing modes. It has 16 general purpose registers. Each register can store 16-bit data. The processor has 16-bit ALU capable of performing 11 arithmetical and logical operations. The processor also incorporates a flag register which indicates carry, zero and parity status of the result. All the modules in the design are coded in Verilog. The individual modules are designed and tested at each level of implementation and finally integrated in a top level module by appropriate mapping. The design entry and synthesis is done using Xilinx ISE 10.1 tool and simulation results are verified using Modelsim 10.2.

Key words— RISC, 16-bit CPU, Verilog

I. INTRODUCTION

When the controller design become more complex in CISC and the performance was also not up to expectations, people started looking on some other alternatives. It had been found that when a processor talks to the memory the speed gets killed. So the one improvement on CPI was to keep the instruction set very simple. Simple in not the way it works but the way it looks. That's why there are very few instructions in any typical RISC architecture where processor asks data from memory probably not other than Load and Store. At the end the pipelining added a new dimension in the speed just with the help of some additional registers, which increases throughput by reducing CPI. Hence the instruction can be executed effectively in one clock cycle[1].

A common misunderstanding of the phrase "Reduced Instruction Set Computer" is the mistaken idea that instructions are simply eliminated, resulting in a smaller set of instructions. In fact, over the years, RISC instruction sets have grown in size and today many of them have a larger set of instructions than many CISC CPUs. The term "Reduced" in that phrase was intended to describe the fact that the amount of work any single instruction accomplishes is reduced at most a single data memory cycle compared to the "complex instructions" of CISC CPUs that may require number of data memory cycles in order to execute a single instruction[2]. Most microprocessors in today's market are based on either RISC or CISC architectures. Research has shown that RISC architecture greatly boosts computer speed by using simplified machine instructions for frequently used functions. The following features typically found in RISC based systems.

1) Pre-fetching: The process of fetching next instruction or instructions into an event queue before the current instruction is complete is called pre-fetching.

2) Pipelining: Pipelining allows issuing an instruction prior to the completion of the currently executing one.

3) Superscalar operation: Superscalar operation refers to a processor that can issue more than one instruction simultaneously.

II. ARCHITECTURE

The objective of the project is to design a 16-bit RISC processor based on Harvard architecture which utilizes minimum functional units. The architecture of proposed 16-bit Processor is shown in Fig.1. The processor incorporates 16-bit ALU capable of performing 11 arithmetical and logical operations, 16-bit program counter, 24-bit Instruction register, Sixteen 16-bit general purpose registers, 3-bit flag register to indicate carry, zero and parity.

The processor has four states idle, fetch, decode and execute. The control unit provides necessary signal interaction to perform expected function in all the states.

The 16-bit program counter indicates the address of memory location from which the instruction is to be fetched. After the execution of current instruction, the program counter is incremented by one unless 'JUMP' instruction is encountered. When the 'JUMP' instruction is executed the program counter is incremented or decremented by the amount indicated by the offset

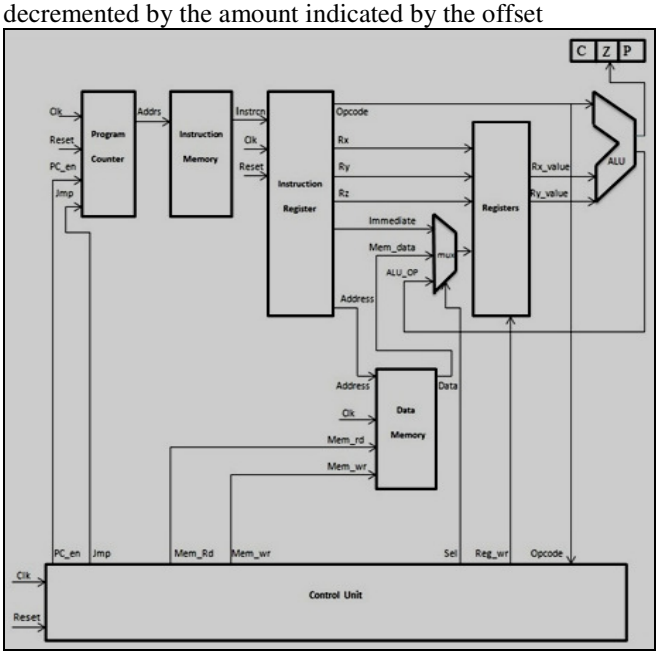

Fig. 1 Architecture

The instructions are written in instruction memory. The output of the program counter is provided to instruction memory as input. The instruction corresponding to the address pointed by the output of the program counter is fetched from instruction memory. The fetched instruction is stored and decoded in instruction register. In the decode process destination register, source register, address of the memory location or immediate value are assigned based on the opcode. The opcode is used by ALU to perform requested operation and also used by control unit to generate necessary control signals.

There are sixteen 16-bit general purpose registers names R0 through R15. The register array has two read and one write ports. When 'reg_wr' signal is enabled, the data is written into a register indicated by the write address. Otherwise two registers indicated by the read addresses are read. The ALU performs 11 arithmetical and logical operations. Some of the operations require two operands, while others need only one. All the operands for ALU operations are provided by registers and the result of operation is written back into the specified destination register through the multiplexer.

Fetch Unit

The function of fetch unit is to obtain an instruction from the instruction memory using current value of the program counter and increment the program counter value for the next instruction. 16-bit program counter and instruction memory together form fetch unit**.**

Decode Unit

The function of the instruction decode unit is to use the 24-bit instruction provided from the previous fetch unit to index the register file and obtain the register data values. The instruction register, control unit and registers together form decode unit.

Execution Unit

The execution unit of the processor consists of arithmetic logic unit (ALU) which performs the operation specified by the opcode.

III. INSTRUCTION SET

The instruction word is 24-bit wide. The first four bits [23-20] specify the opcode. The next 4-bits [19-16] represent destination register. Remaining 16-bts [15-0] represents either immediate value or source registers Rx and Ry or address location depending upon the type of instruction.

Operation	Opcode				Result
HLT (0)	0	0	0	0	STOP
ADD (1)	0	0	0	1	Rz=Rx + Ry
SUB (2)	0	0	1	0	Rz=Rx - Ry
MUL(3)	0	0	1	1	Rz=Rx * Ry
AND (4)	0	1	0	0	Rz=Rx & Ry
OR (5)	0	1	0	1	Rz=Rx | Ry
XOR (6)	0	1	1	0	Rz= Rx ^ Ry
NOT (7)	0	1	1	1	Rz= ~Rx
SHL (8)	1	0	0	0	Shift Left Rx
SHR (9)	1	0	0	1	Shift Right Rx
INC (a)	1	0	1	0	Increment Rx
DEC (b)	1	0	1	1	Decrement Rx
MVI (c)	1	1	0	0	Move Immediate
LOAD (d)	1	1	0	1	Read from memory
STORE (e)	1	1	1	0	Write to memory
JUMP (f)	1	1	1	1	Jump to target address

Table 1 Instruction Set

The instructions can be classified into the following five functional categories: data transfer operations, arithmetic operations, logical operations, branching operations and control operations.

Data Transfer Operations
This group of instructions copies data from a location called source to another location called destination without modifying the contents of the source. MVI, LOAD and STORE instructions come under this category.

Arithmetic Operations
These instructions perform arithmetic operations such as addition (ADD), subtraction (SUB), multiplication (MUL), increment (INC) and decrement (DEC).

Logical Operations
These instructions perform various logical operations such as AND, OR, XOR, NOT, SHL, SHR

Branching Operations
The instruction JUMP alters the sequence of program execution

Control Operations
The instruction HLT does not produce any result but stops the program execution.

IV. FSM OF THE PROCESSOR

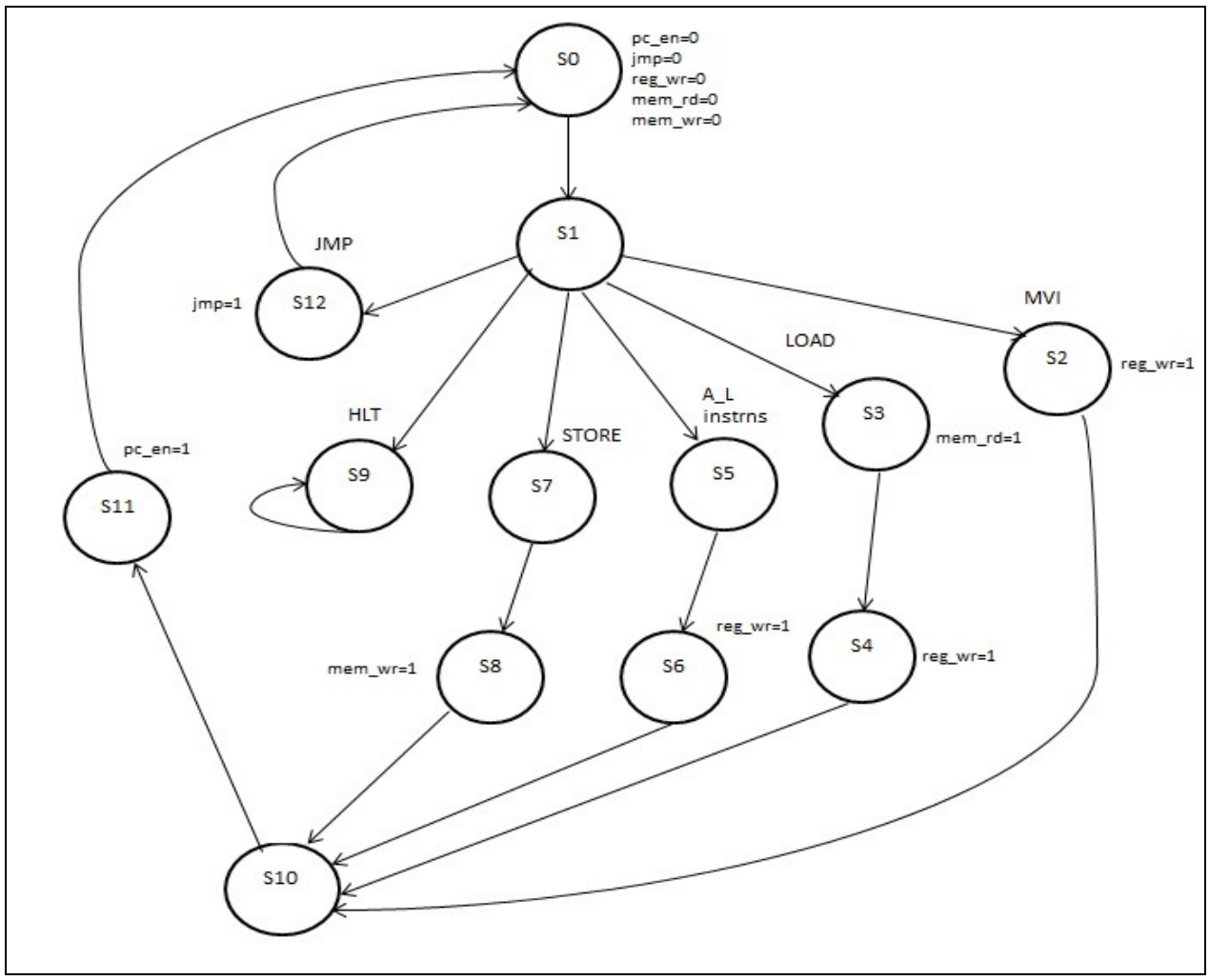

Fig. 2 Finite State Machine

When the 'reset' signal is high, the processor will be in the idle state which is indicated by S0. In S0 state all the control signals 'pc_en', 'jmp', 'reg_wr', 'mem_rd', and 'mem_wr' are low. The program counter points to the first address of

instruction memory 0000H. When the reset signal goes low, the first instruction stored at 0000H is fetched. Now the processor enters S1 state. This state controls the decode operation.

If the decoded instruction is MVI, the processor enters S2 state. The MVI instruction commands the processor to store immediate value into the specified register. Since in this case a register has to be written, the control signal 'reg_wr' is made high by the control unit.

If it is a LOAD instruction, it requires data to be loaded from memory into a register. Hence in the first phase of execution i.e. at state S3, 'mem_rd' signal is made high. This allows the processor to read the contents of memory location. In the next phase of execution i.e. at state S4, the data read from data memory is to be written into a register. Hence 'reg_wr' signal is enabled at state S4.

If the instruction is one of the ALU instructions, the processor enters state S5. Since ALU instruction requires operands to be fetched from registers, 'reg_wr' signal is made low. This allows the processor to read register contents. In the next phase the result of operation should be written back into the registers. Hence 'reg_wr' signal is made high at S6.

If it is a STORE instruction, in the first phase register contents are read while 'reg_wr' signal is low at S7. In the next phase the data read from the register should be written in a memory location. Hence 'mem_wr' signal is made high at state S8.

After execution of all the instructions except JUMP and HLT, the processor goes to state S10 which provides one cycle delay. This delay is necessary for proper synchronization. In the next state S11, 'pc_en' signal is made high which enables the program counter to generate next sequential address.

If the decoded instruction is JUMP, the processor enters state S12. In this state control signal 'jmp' is made high. This stops the program counter from generating next sequential address and updates program counter with the target address.

When the HLT instruction is executed the processor enters S9 state and does not come out of that state. Further execution of instructions is stopped.

V. SIMULATION RESULTS

MVI

The fig. 3 shows the simulation results for MVI instruction. The instruction MVI R1 0005 is written at address 0000h of instruction memory. In the decode process destination register 'Rz' is assigned with R1 and 'immediate_value' is assigned with 0005. At the next positive edge of the clock cycle when 'reg_wr' signal goes high, the value 0005 indicated by 'reg_wr_data' is written into the register R1.

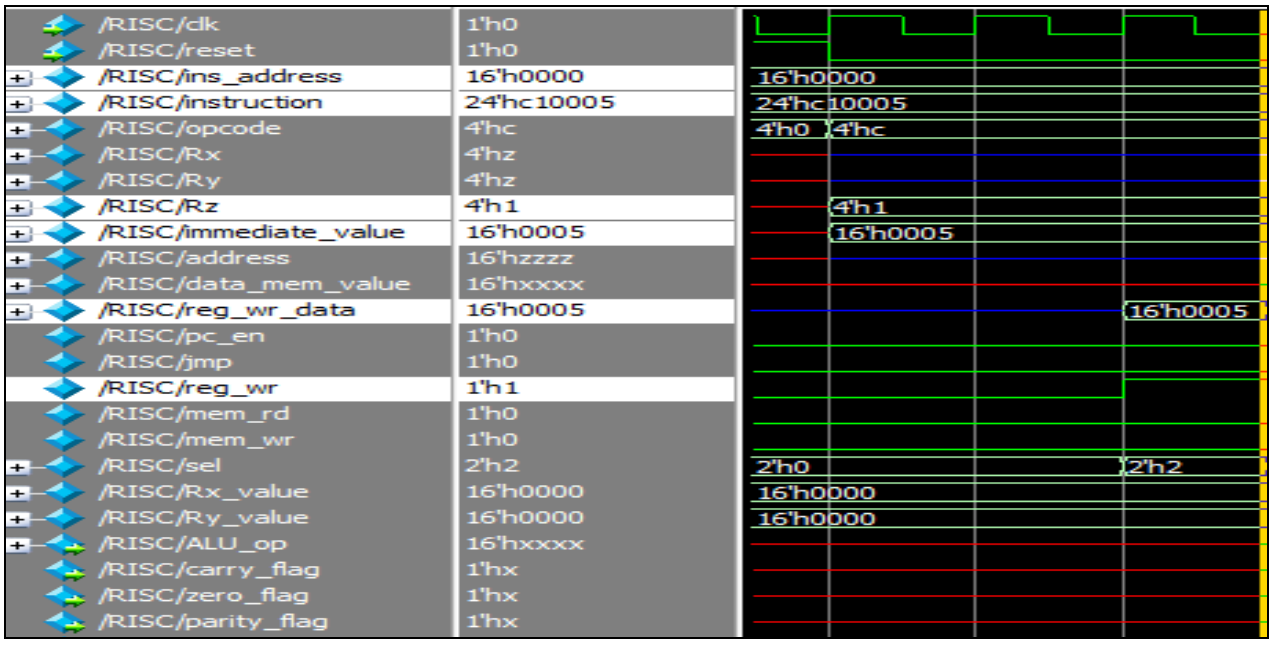

Fig. 3 MVI

LOAD

The instruction LOAD R2 0004 is written at address 0001h of instruction memory. In the decode process destination register 'Rz' is assigned with R2 and 'address' is assigned with 0004. At the next positive edge of the clock cycle when 'mem_rd' signal goes high, the contents of data memory at address 0004h is read and found to be 0003. In the next clock cycle when 'reg_wr' signal goes high, the value 0003 indicated by 'reg_wr_data' is written into the register R2.

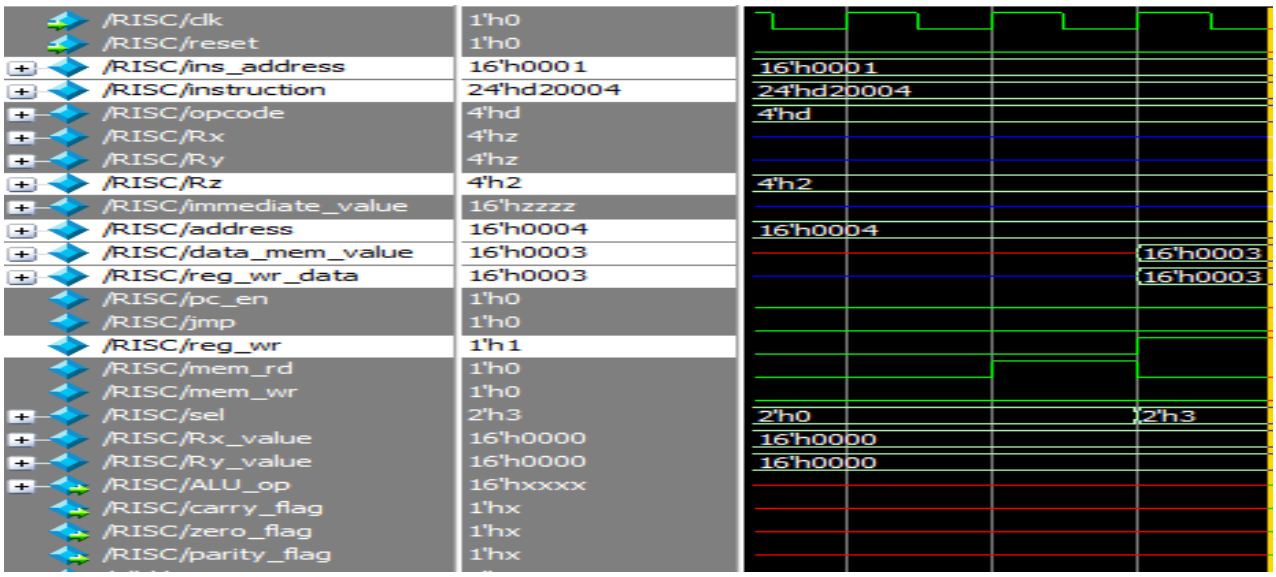

Fig. 4 LOAD

ADD, SUB, MUL

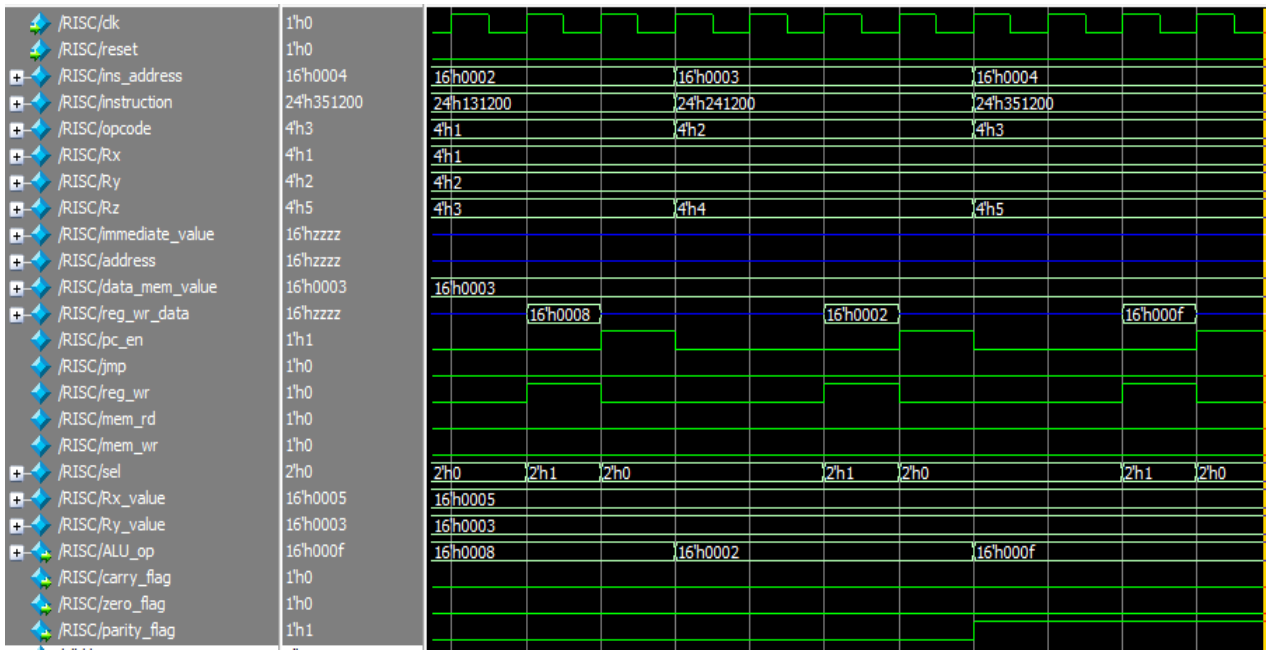

Fig. 5 ADD, SUB, MUL

When the instruction ADD R3 R1 R2 is executed, R1 and R2 values are retrieved, addition operation is performed and the result 0008 is stored in register R3. Similarly subtraction and multiplication operations produce results 0002 and 000f which are stored in R4 and R5 respectively. The result of multiplication operation produces the value 000f. Since the number of 1s in the result is even, parity flag goes high.

Control Signals

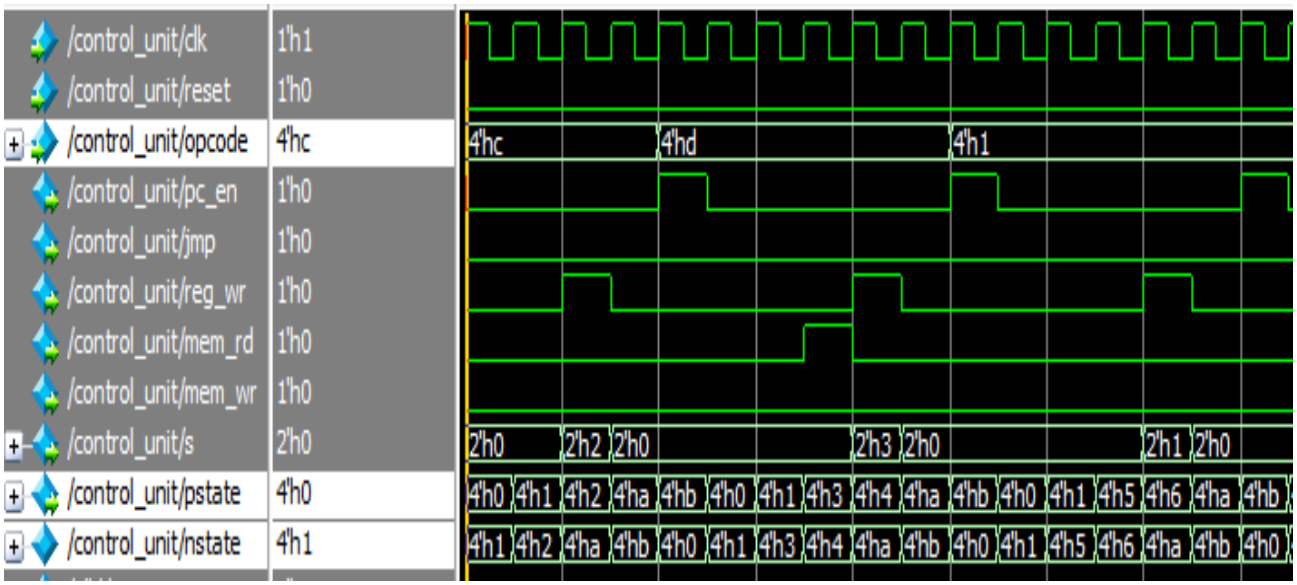

Fig. 6 Control Signals

VI. CONCLUSION AND FUTURE WORK

A 16-bit RISC processor has been designed that utilizes minimum functional units. The design is based on Harvard architecture. The design entry and synthesis is done using Xilinx ISE 10.1 tool. From synthesis report it is found that the minimum clock period that can be achieved using the design is 14.95ns. Simulation is done using Modelsim 10.2 simulator. The simulation output is compared with the expected results and the functionality is found correct.

The design can be improved in number of ways. To achieve a more sophisticated design more features can be added to the current design. The number of instructions that the processor supports can be increased. Pipelining can be added to improve the performance of the proposed design.

REFERENCES

[1]. Mamun B, Shabiul I. and Sulaiman S,"A Single Clock Cycle MIPS RISC Processor Design using VHDL"

[2]. Hamblen J." Synthesis, Simulation, and Hardware Emulation to Prototype a Pipelined RISC Computer System"

[3]. Zainalabedin N,"VHDL for Modeling and Design of Processing Units"

[4]. Takanori M, Satoshi A and Masaaki I, "A Multithread Processor Architecture Based on the Continuation"

[5]. Kasuga-Koen, Kasuga, Fukuoka, " The Innovative Architecture for Future Generation High-Performance Processors and Systems"

[6]. Virendra S. and Michiko I,"Instruction-Based SelfTesting of Delay Faults in Pipelined Processors", IEEE Transaction on VLSI systems, vol. 14, no.11,pp.1203-1215.

[7]. Patterson A. and Hennessy J,"Computer Organization & Design", Morgan Kaufmann Publishers, 1999

[8]. Peter J Ashenden,"Digital Design, An embedded systems approach using Verilog", Morgan Kaufmann Publishers,2010

[9]. Ramesh Gaonkar,"Microprocessor architecture, programming and applications with the 8085", Penram International Publishing,1989

