Section: Research Paper

Design and Implementation of Program and Loading unit using Double Precision Floating-Point Operation for RISC Architecture

D.Divya[1], A.Poorinma[2] I.Vivek Anand[3]

[1,2] Student, ECE department, National Engineering College, Kovilpatti, India
E-Mail: divyad1898@gmail.com , 1911012@nec.edu.in
[3] Assistant Professor (Senior Grade), ECE Department, National Engineering College, Kovilpatti, India
E-Mail: ilangovivek@gmail.com

Abstract:

In recent days, reduced instructions play a vital role in the processor development. Reduced Instruction Set Computers (RISCs) are such instructions which provide offers to these applications. This project suggests using double precision to implement the programme and loading unit on 64-bit RISC processors. The characteristics of this processor are they operate at fast speed while using low power. The CPU is divided into three sections: the execution portion, the instruction decodes and fetch. By employing parallel architectures, floating point ALU with double-precision is proposed. The proposed architecture not only improves the speed, but also the precision. Design of each and every block is written using Verilog HDL and simulation results are observed.

Keywords: Floating Point, Delay, Instruction Fetch

Introduction:

Digital signal processing applications rely on floating point operations [1-2]. Subtraction and Addition operations are considered as a fundamental one in floating-point arithmetic. In order to obtain a high precision operation, instead of taking a static operation, dynamic range is considered. Arithmetic unit implementation is made possible with Field Programmable Gate Arrays and these devices not offers low cost but also results in high performance and package density.

Accessible technology is the preferred choice in design. Technology leads to the development of hardware, software, and alternate implementations. Additionally, CPUs and double precision floating point have been created better. RISC architecture has an impact on emerging technologies and aggregates CISC designs' philosophical ideas. Small and slow memories were the primary source of processor problems. To address this tendency, RISC processors were developed, which heavily rely on software with scan addressing modes. The other key characteristics of RISC included its high register density, single clock cycle execution, load/store operations required to access memory, and ease of pipelining [3]. Due to ongoing development, VLSI applications were growing radically complicated and varied. Digital signal and speech processing needed higher degrees of precision, dynamic range, and speed to improve. The sole method to prevent bugging is the floating-point arithmetic unit. This led to the parallel integration of the floating-point and fixed-point arithmetic units.

The advantages of this processor were its high speed and low power consumption. The RISC processor, which has the ability to speed up individual instructions and provide a net performance improvement, is the foundation for creating high performance processors. As a

result, RISC processors used many fewer transistors than CISC processors, which led to them taking up less space overall. In order to execute a single clock operation, load and store instructions are preferred which uses a access memory rather that IO instructions [4-6]. Less instructions are used in the instruction set, which results in optimised compilers [7-8]. This idea is used in many embedded and portable applications.

Floating Point Arithmetic using Double Precision:

Floating point representation is used for representation a number either large or in small values. Compared to fixed-point, floating-point exhibits better resolution and accuracy. The sign, mantissa, and exponent all serve as representations of floating-point numbers. Fig.1 shows the double precision format. Starting from left to right, numbers from 0 to 63 are needed for representing word of 64-bit size.

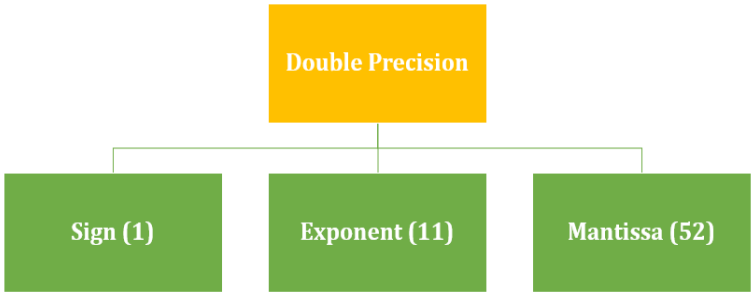

Fig.1. Double Precision Format

Floating point numbers are represented by collection of integers named as double precision. When compared to single precision, double precision produces more accurate results. In contrast to single precision, the number of words employed is double and hence it is named as double precision. As an example, if a 32-bit integer is needed for single precision, a 64-bit number is necessary for double precision.

Proposed RISC Architecture:

Less transistors are needed for RISC devices, which lowers the cost of design and production. Simple and fundamental instructions are part of the RISC instruction set, from which more complicated instructions are frequently derived. It has a limited number of instructions in a predetermined format. Between registers, there is data transfer. Register-based instruction types are required. A single clock cycle is all that is required for a brief memory access.

Three stages are involved in processor: execution, instruction fetch, and instruction decoding. To improve the performance and execution, an architecture is created in the ALU that employs double precision and improves execution significantly. In this work, the mantissa and exponent bits are increased in comparison to single precision to create the double-precision floating-point ALU.

Representation of RISC Architecture using Double Precision:

The following Fig. 2 shows a RISC-based architecture featuring a double-precision floating point ALU. The proposed method is composed of eight primary building blocks: instruction decoder, fetch, execute unit and a double precision floating point, control and memory unit, data, instruction memory and register bank. The RISC architecture was created by building and connecting the aforementioned building blocks.

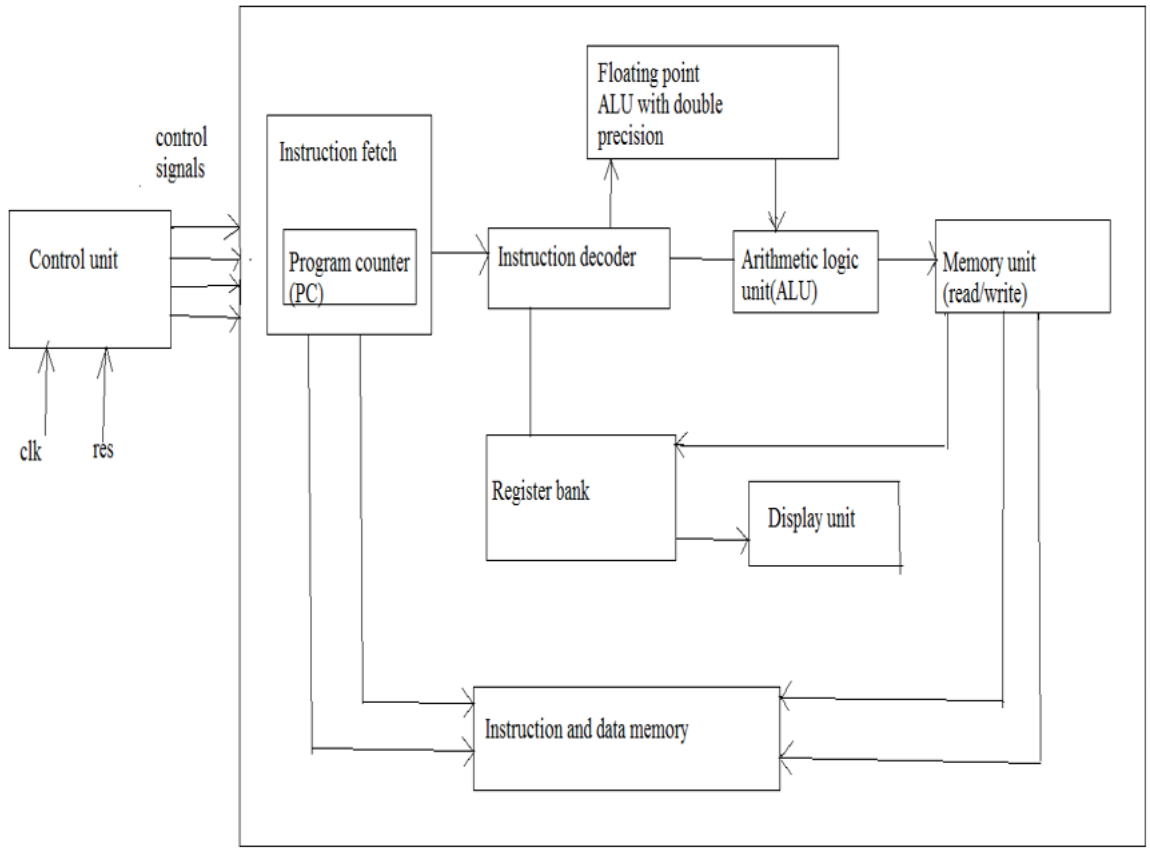

Fig. 2 Double-precision floating point ALU in a RISC-based architecture

- **Instruction Fetch Unit:**

The initial stage of a MIPS RISC is the instruction fetch stage. During the instruction fetch stage, the desired instruction is fetched from memory. To start the operation of the instruction fetch stage, the program counter, a 64-bit register, is sent out to fetch the instruction from memory into an instruction register, and the (PC) is then increased by 4 through an adder to address the subsequent sequential instruction.

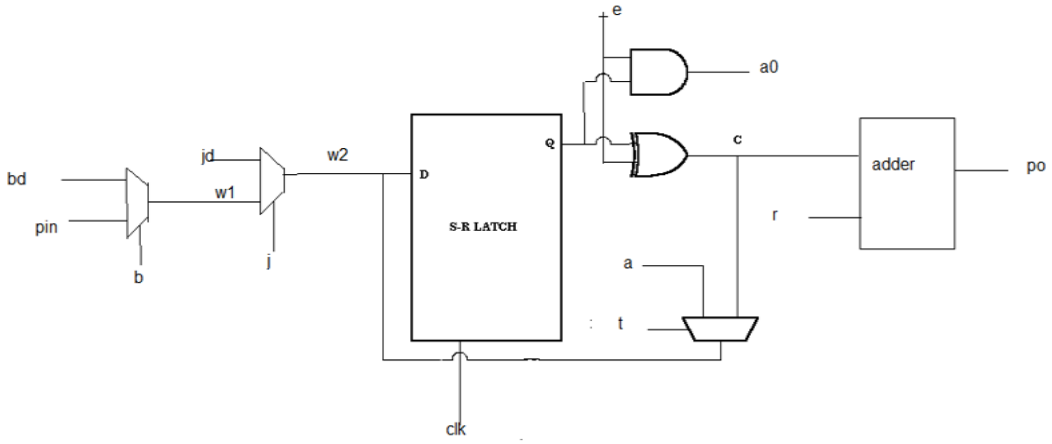

Fig. 3 Instruction Fetch Architecture

The instruction register stores the instruction that will be needed on subsequent clock cycles. The branch instruction consists of three operands, two registers that can be compared for equality, and a 16-bit offset. In relation to the branch instruction address, the

branch target address is calculated. The instruction's sign-extended offset field is then added to the PC in order to determine the branch target address.

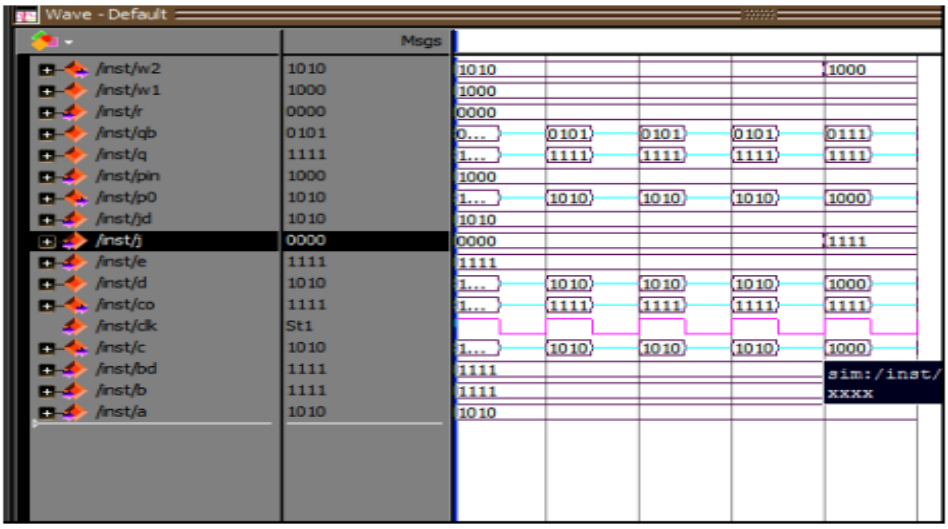

Fig. 4 Simulation result of Instruction Fetch unit

Fig. 4 illustrates pin, jd, j (select line), Bd, select line is taken as b, clk, t, e, and r are the inputs. The output of flip-flop is labelled c, c0, q, qb, and p0, while the program counter output pins are c and c0. The final input from instruction fetch is labelled p0.

- **Arithmetic and Logical Block:**

The ALU includes logical, relative, shifting, and arithmetic operations and is sometimes known as an execution unit [5]. The four fundamental operations in mathematics are subtraction, multiplication, addition and division. Apart from these operations comparison operations are also performed. Rotations to the right, left, right, and left are a few instances of shifting operations. Then, there are logical operations like XOR, NOR, OR, AND, NAND, and XNOR available.

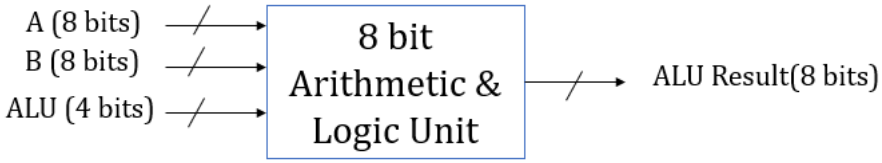

Fig. 5 8 bit Arithmetic and Logical Unit

A and B are the inputs in Fig. 5, and ALU select is the select line that determines which operation will be carried out. The output pin then contained the ALU block's result.

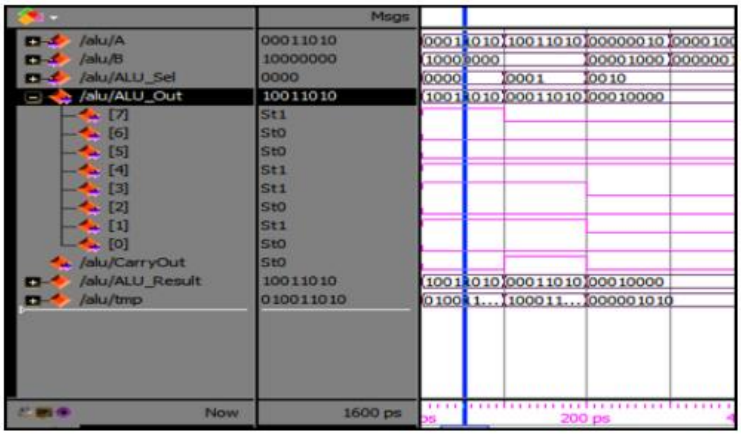

Fig. 6 Simulation result of 8 bit ALU

In fig. 6 the inputs are considered as ALU_In, B and ALU-Sel and the outputs are observed in the variables ALU result and ALU-Out

- **Double Precision Floating-Point Addition:**

There are three main steps, as depicted in fig. 7. 1) In the case of double precision arithmetic, 1 bit value is considered as sign, exponential takes 11 bits and 52 bits are assigned as mantissa components

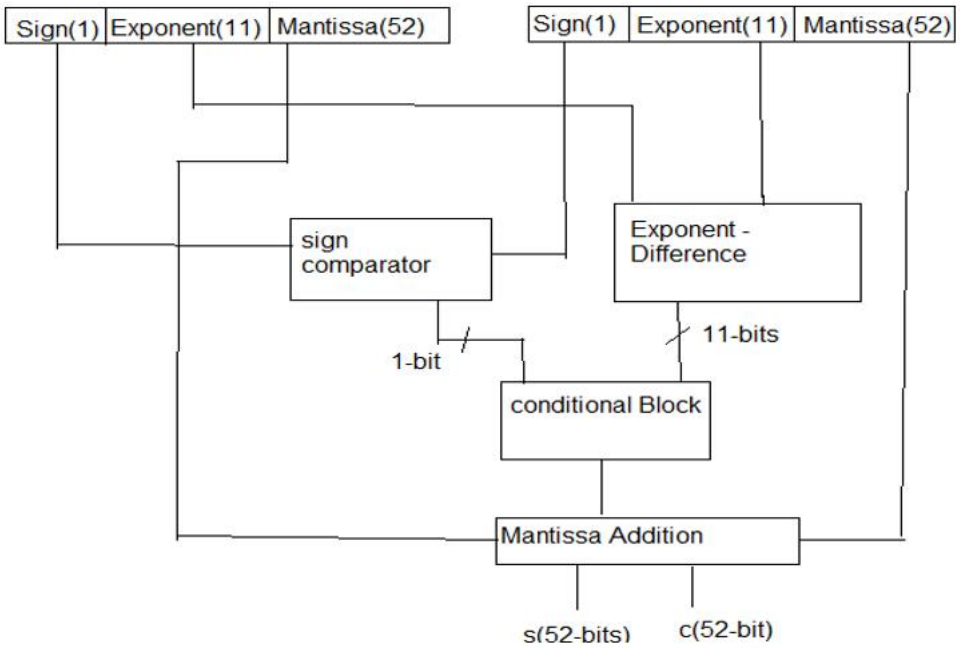

Fig.7. Architecture of Double Precision Floating-point Addition

- **Architecture of Double Precision Floating-point Addition:**

The conditional block used to highlight the results of addition and subtraction of floating-point terms. These steps compare the sign and exponent of the inputs provided in the first step and second step, respectively. The mantissa component is then subtracted as the third step.

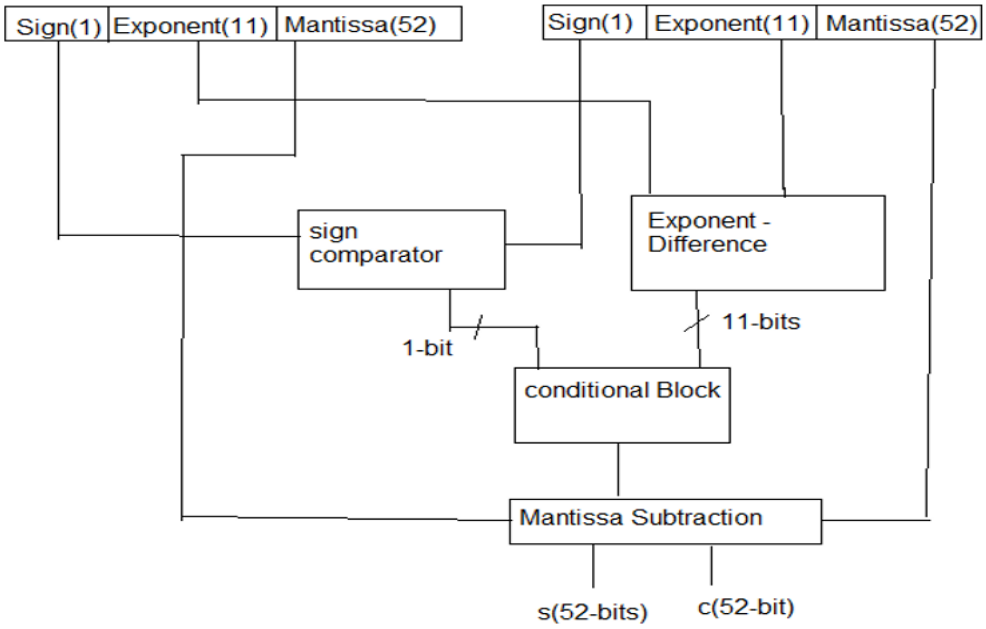

Fig.8 Architecture of Floating-point Subtraction using Double Precision

The outputs obtained for double precision arithmetic for addition and subtraction are depicted in Fig. 9 and 10.

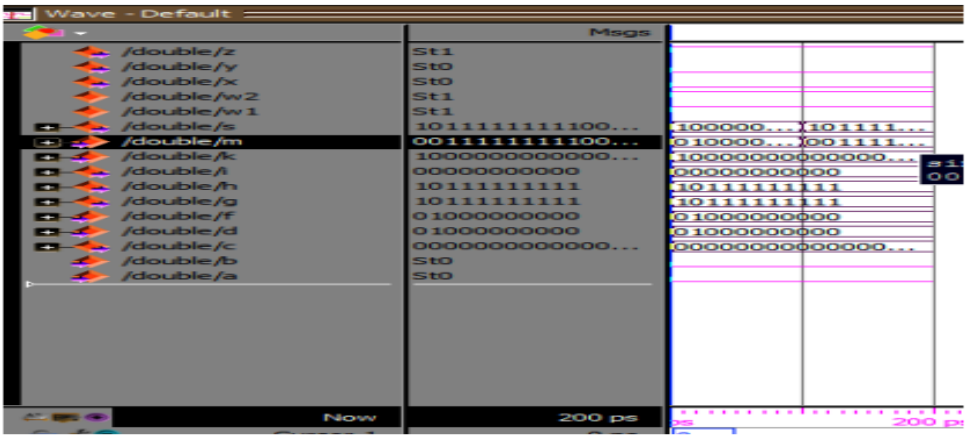

Fig. 9 Simulation result of Addition operation performed using double precision

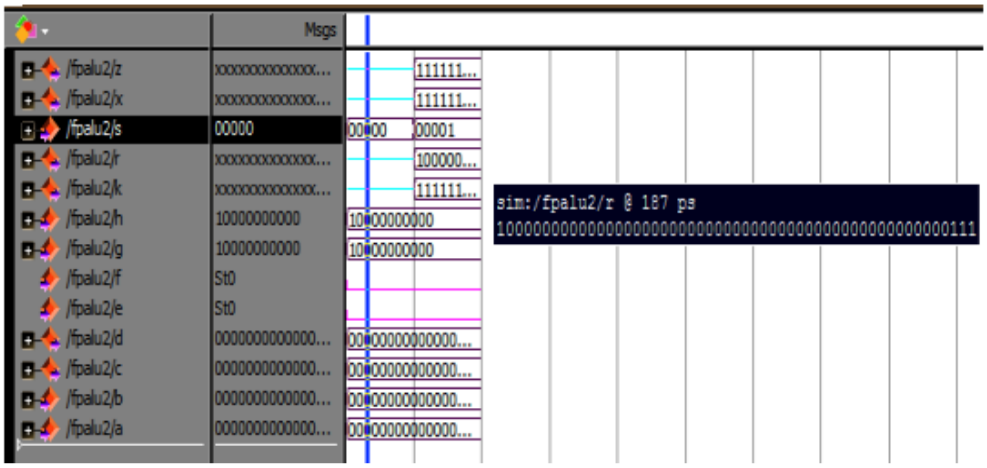

Fig.10 Simulation result of Subtraction operation performed using double precision

The simulated output of a ROM utilizing the inputs address clk and read_en is shown in Fig. 11 below. Based on the address selection, when read_en =1, the data is reflected at the output. The result displays the value that has already been saved in the designated location.

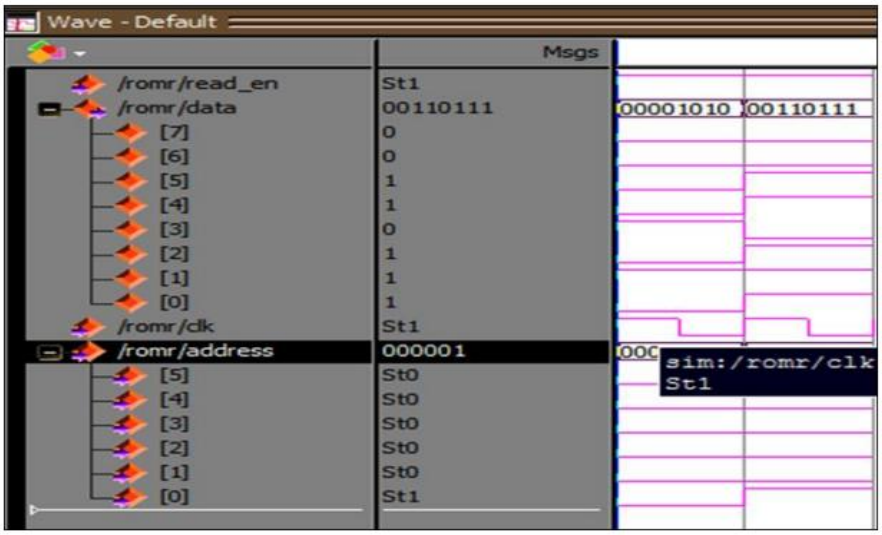

Fig. 11 ROM Memory Simulation Output

- **Program Counter:**

The program counter holds the next instruction to be executed. The previous values get updated as the program counter increments and this is accomplished when the instructions are fetched from memory.

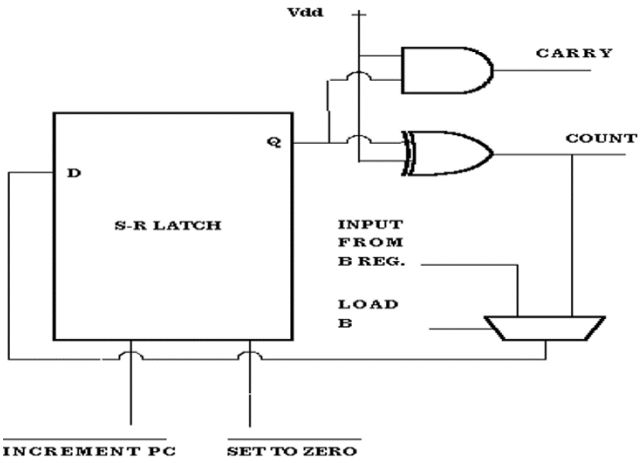

Fig. 12 Block diagram representing the program counter

When the sufficient instructions are reached, the program counter revised the next instructions and when the machine are reset, program counter initializes to zero. The program counter, which is depicted in Fig. 13, stores address and aids the instruction fetch block in retrieving instructions before passing them to the next level, the instruction decoder.

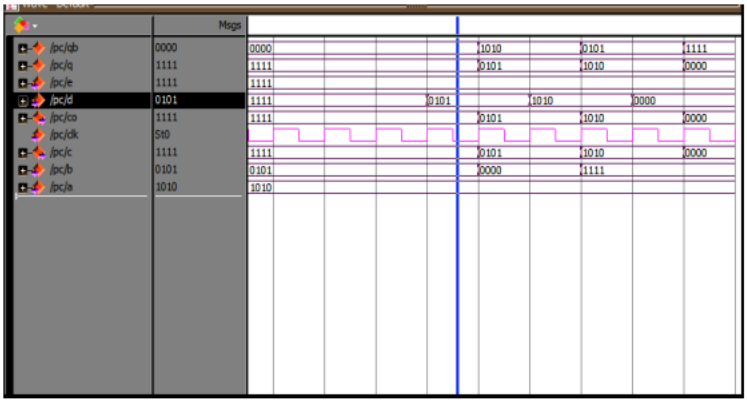

Fig. 13 Program Counter Simulation Output

- **Storage Unit:**

The 2^6 memory spaces required for a system are contained in a storage area known as a memory unit. The floating point unit is used to save the results produced from the ALU and memory interface.

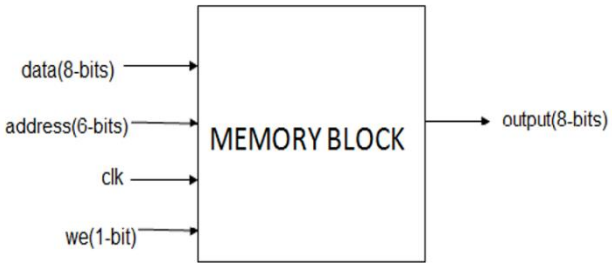

Fig. 14 Representation of Storage Unit

In fig. 14 Data, address, clk, and we (write-enable) are the inputs. The information is output from additional blocks, such as Floating-pint ALU or ALU. The output from these blocks was supplied as input to the memory block when the necessary operation had been completed. The address pin indicates the location where the data must be stored. The write enable pin, which helps to write the data in the necessary location, is then we (a single bit).

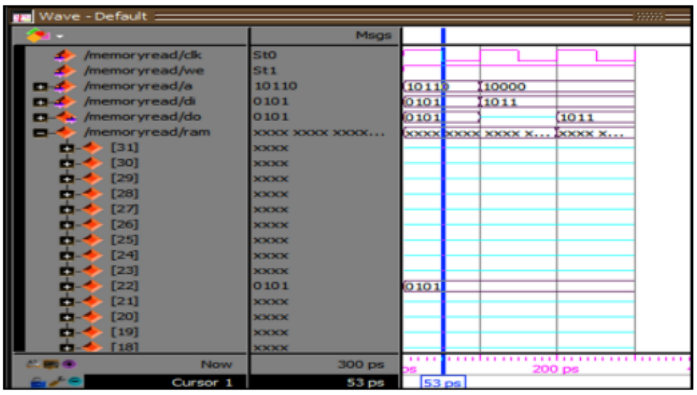

Fig. 15 Memory Block Simulation result

- **Interfacing Memory with ALU Unit:**

The next step is integrating the memory and ALU unit. In this case, the memory unit and ALU architectures are taken from Fig. 6 and 14 and its corresponding simulation result is shown in figure 16.

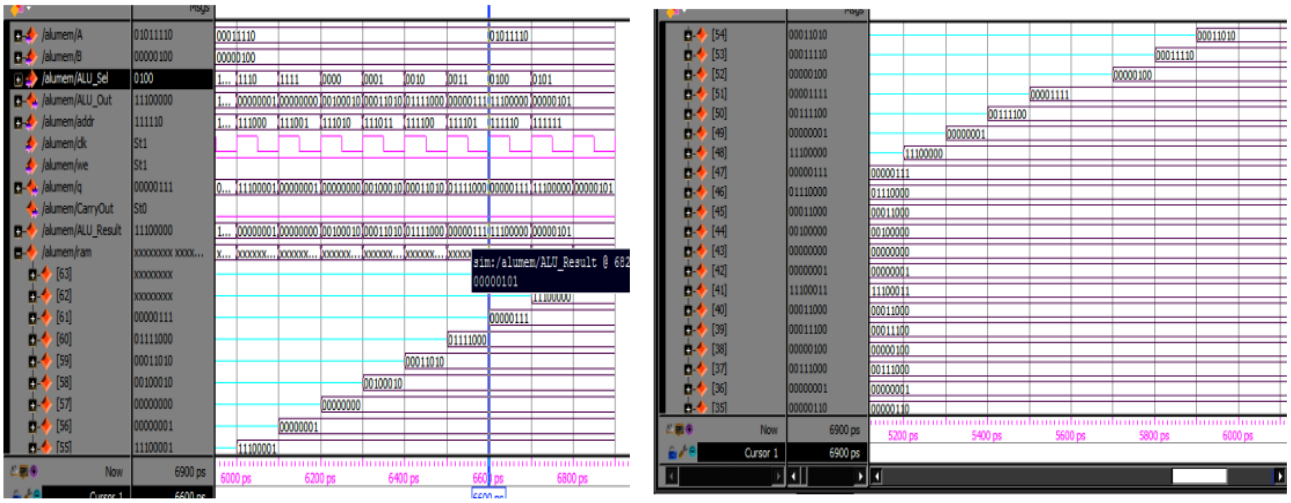

Fig. 16 Simulation result of Memory and ALU interface

- **Register Bank:**

In order to access the programmable registers, assembly language is preferred in register bank and is shown in fig.17. It is considered to be a software array's physical equivalent. In register bank, specific index is allocated for writing and reading the data. The caches and main memory are used for writing and reading the data.

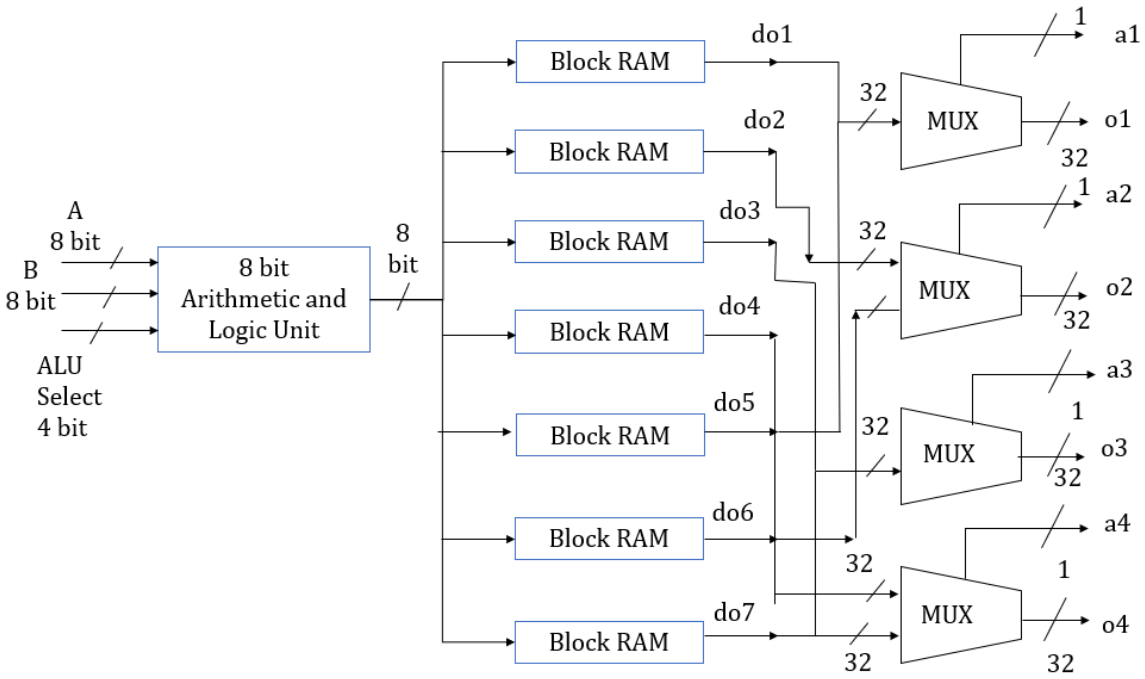

Fig. 17 Block diagram of Register Bank architecture

The address and data were provided as input, and as shown in fig. 11.9.1, they were processed and provided as output.

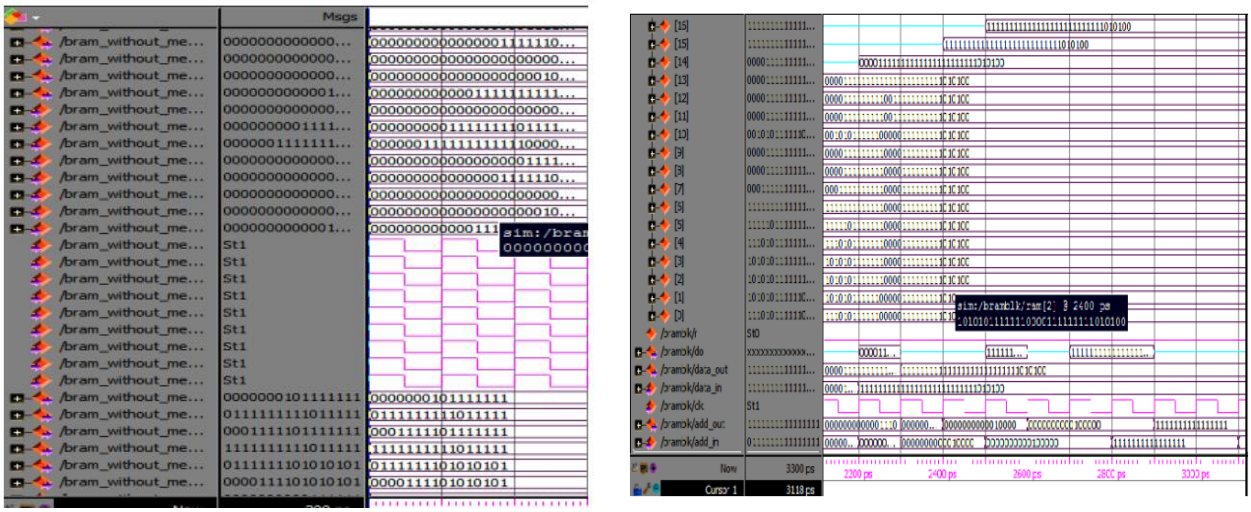

Fig. 18 Register Bank Simulation result

- **Programming Unit:**

  The following fig. 19 shows the programming unit proposed for RISC processor.

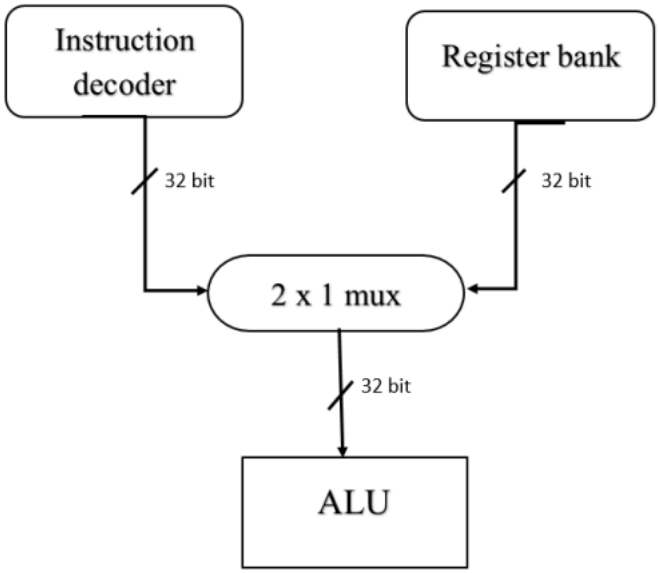

Fig. 19 Block diagram of programming unit

The ALU takes the value of register bank and instruction decoder. 2x1 mux is used to select the values of these two. In any index the values are read, by the 32 bit register bank. There are separate interfaces available for accessing main memory and cache memory.

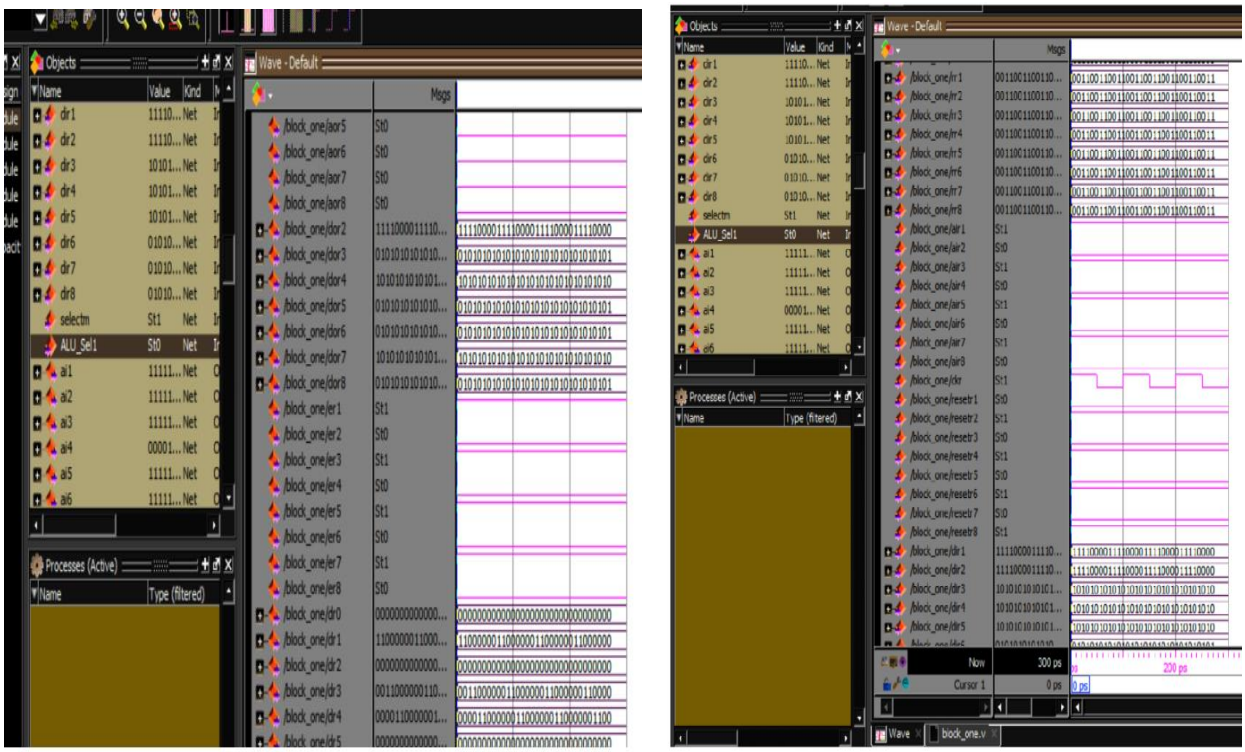

Fig. 20 Simulation result of Programming Unit

- **Loading Unit:**

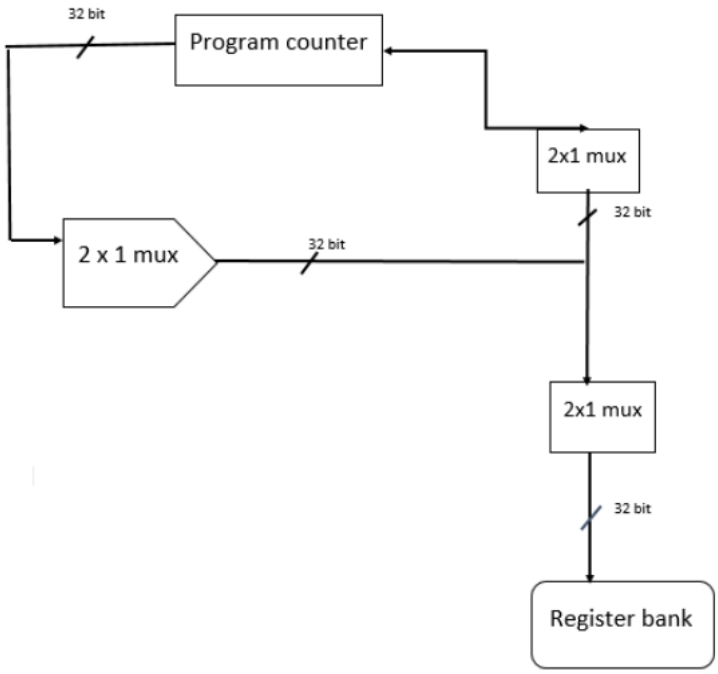

Fig. 21 Block diagram of Loading Unit

The program counter stores a value that are executed by the computer in a specified address. 2x1 multiplexers get the output from the program counter as input, and a third 2x1 multiplexer receives the output from the multiplexer as input. The register bank receives the multiplexer's final output.

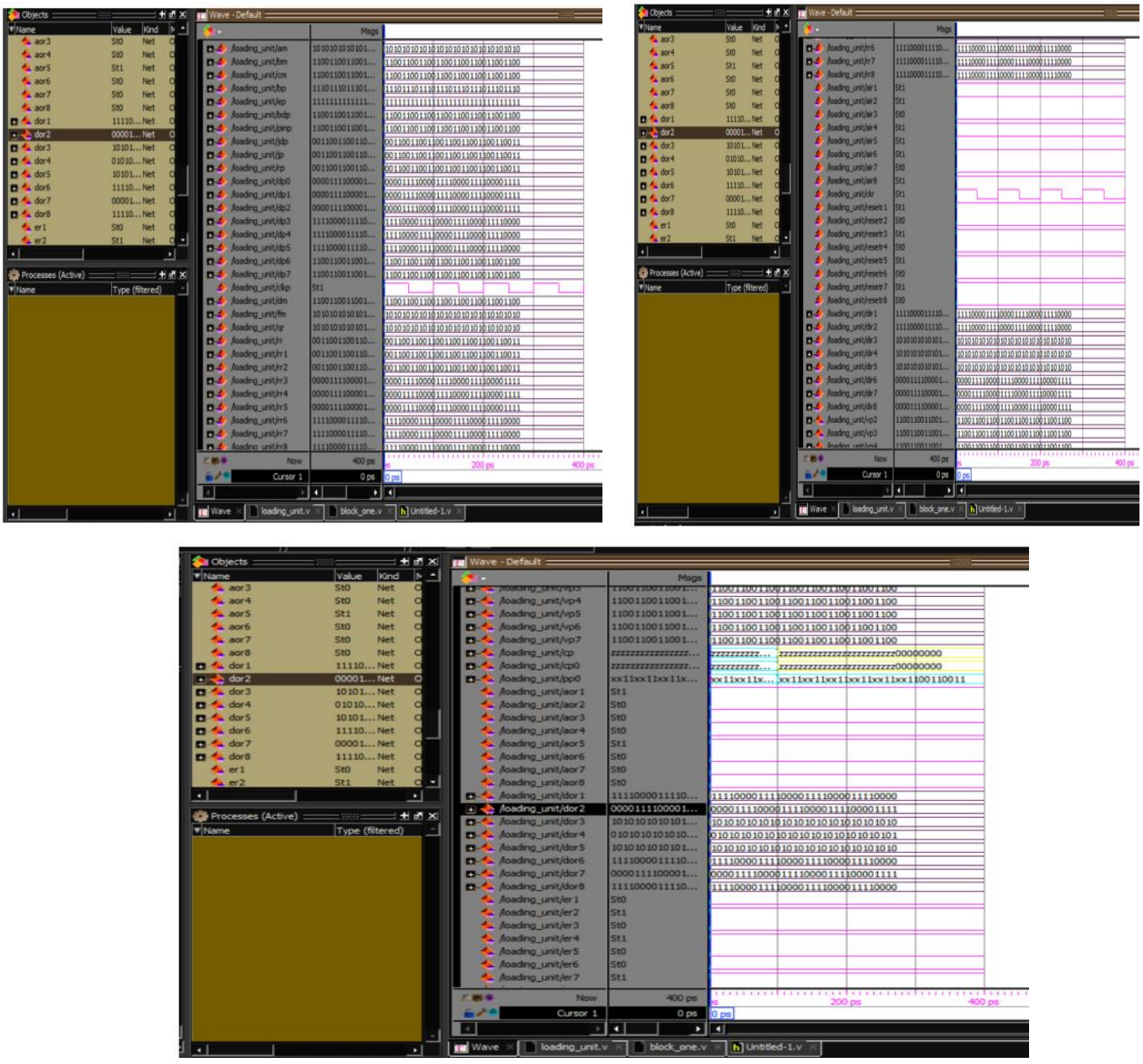

Fig.22 Simulation Result of Loading unit

- **Overall RISC Architecture:**

Fig. 23 displays the simulation results for a RISC architecture with the proposed architecture. The outcome of the process is saved in the register bank at the designated location. The final simulated result is displayed in fig. 23 after the suggested loading and programming unit is cascaded in the RISC processor.

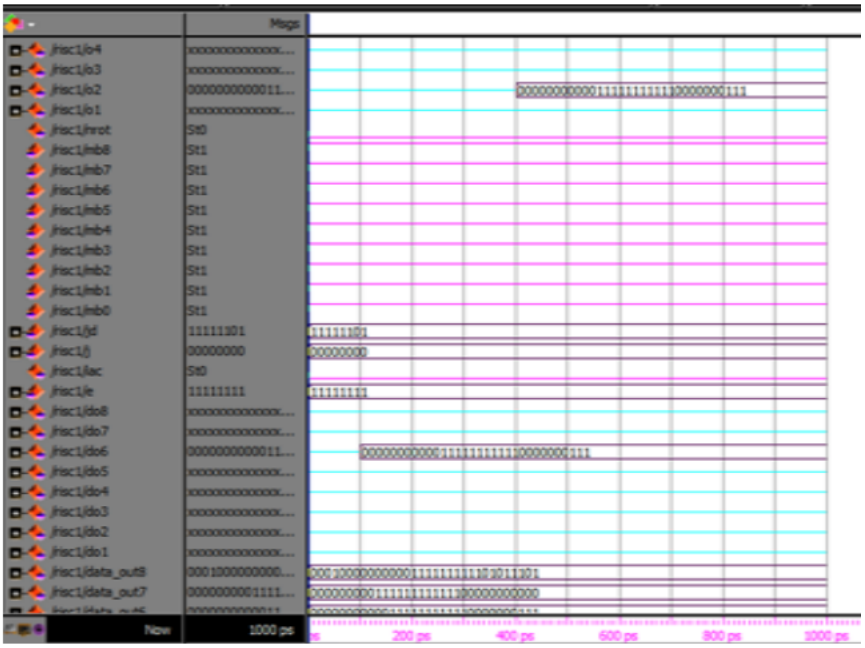

Fig. 23 RISC Architecture simulation result

- **Conclusion and Future Work:**

In this work, a RISC processor is designed by creating a double precision floating point ALU arithmetic. Suitable register banks are taken for implementing the function. The proposed architecture not only reduces the occupancy but also decreases latency and power. In future, it is planned to extend this architecture to 128 bits.

References:

[1] Grover, N., & Soni, M. K. Design of FPGA based 32-bit Floating Point Arithmetic Unit and verification of its VHDL code using MATLAB. International Journal of Information Engineering and Electronic Business,6(1), 1(2014).

[2] Divya. D, Balasaraswathi, R, Harini kalyani M,Vivek Anand. I. "Modeling and Execution of Floating-Point Parallel Processing Operation for RISC Processor", International Journal of Engineering and Advanced Technology, 2020

[3] Ritpurkar, S. P., Thakare, M. N., & Korde, G. D. Design and simulation of 32- Bit RISC architecture based on MIPS using VHDL. International Conference on Advanced Computing and Communication Systems (pp. 1-6). IEEE,2015- January.

[4] Mane, P. S., Gupta, I., & Vasantha, M. K. Implementation of RISC Processor on FPGA. IEEE International Conference on Industrial Technology (pp. 2096-2100). IEEE,2006- December.

[5] Paldurai, K., & Hariharan, K. FPGA implementation of delay optimized single precision floating point multiplier. International Conference on Advanced Computing and Communication Systems (pp. 1-5). IEEE, 2015- January.

[6] Kumar, J. V., Swapna, C., Nagaraju, B., & Ramanjappa, TFPGA Based Implementation of Pipelined 32-bit RISC Processor with Floating Point Unit. IEEE International Conference on Communication and Signal Processing, 2014.

[7] Gollamudi, P. S., & Kamaraju, M. Design of High-Performance IEEE- 754 Single Precision (32 bit) Floating Point Adder Using VHDL. International Journal of Engineering Research & Technology, 2(7), 2264-2275, 2013.

[8] Tomar, A. K. S., & Jain, R. 20-Bit RISC and DSP System Design in an FPGA. Computing in Science & Engineering, 16(2), 16-20, (2013)