NVEO
Natural Volatiles & Essential Oils

Design of pipelined MIPS Processor with Cache controller using Verilog Implementation

Deepika R [1], Gopika Priyadharsini S M [2], Malini Praba M [3], Muthu Malar M[4], Vivek Anand I[5]

[1]Student, Department of Electronics and Communication Engineering, National Engineering College, Kovilpatti, Tamil Nadu, India;deepikarajaram128@gmail.com

[2]Student, Department of Electronics and Communication Engineering, National Engineering College, Kovilpatti, Tamil Nadu, India;gopikapriyadharsini06@gmail.com

[3]Student, Department of Electronics and Communication Engineering, National Engineering College, Kovilpatti, Tamil Nadu, India; ammu02092000@gmail.com

[4]Student, Department of Electronics and Communication Engineering, National Engineering College, Kovilpatti, Tamil Nadu, India;pearlflower0905@gmail.com

[5]Assistant Professor, Department of Electronics and Communication Engineering, National Engineering College, Kovilpatti, Tamil Nadu, India;ilangovivek@gmail.com

Abstract

Abstract

The advance of VLSI technology has been the enhancing feature in the appearance of VLSI circuits handling floating point (FP) arithmetic. The different requirements for various processor applications also differ, i.e., some processors have a rich repertoire of functions but results in low performance, while some processors aim at having the highest throughput but use more operations such as multiply and add which produces more latency. For real-time processing requirements, performing a large amount of FP operations are considered as a major bottleneck due to the excessively long run time required. The FP arithmetic typically requires additional operations such as alignment, normalization and rounding, giving rise to some significant increase in terms of area, power consumption and computational latency. Such a problem might be mitigated by employing the fused FP add-subtract and dot-product units specially designed for those dedicated computing-intensive tasks. Rounding concentrates on the critical path and high-speed rounding algorithms are used to increase the performance for floating-point multiplication.

To achieve high performance with minimum increase in hardware, existing rounding algorithms like mantissa, exponent and sign are used to generate two consecutive values in parallel, and compute the rounded product by using these values. In this paper floating point ALU with double precision architecture with register bank for implementing RISC processor will be presented. Floating point arithmetic unit using double precision is planned to construct.

In this paper, the instruction cache and data cache will be taken separately and located in the CPU (Central Processing Unit) core. Write back policy is planned to be used where no replacement algorithm is required. After completing the cache controller design, it will be combined with a pipelined MIPS processor and used in programs execution. Thus, the proposed double precision unit will be used as one of the modules for designing an pipelined MIPS processor. The circuits are designed by Encounter RTL (digital design) using Cadence and the simulation results will be observed using spectre tool.

Keywords: MIPS processor, floating point, cache

Introduction

A typical memory hierarchy begins with the cache, a small, expensive, and relatively quick unit, followed by the main memory unit, which is larger, less expensive, and relatively slower. Programmers would demand endless amounts of rapid memory, as computer pioneers predicted. A memory hierarchy, which takes use of locality and trade-offs in the cost-performance of memory technologies, is a cost-effective answer to that objective. According to the principle of locality, most programmes do not access all code or data in the same way. Locality exists in both time and space. Many earlier studies have used Verilog to create a pipelined MIPS (Microprocessor without Interlocked Pipeline Stages) processor with a main ideal memory that can be accessed in a single clock cycle [3-5]. However, this would only be true if the memory was extremely limited or the processor was extremely slow.Because Verilog code is one of the ways for

describing a hardware design, it was employed in this study to build the cache controller architecture for the pipelined MIPS processor.

Proposed Architecture of MIPS Processor with Cache Controller

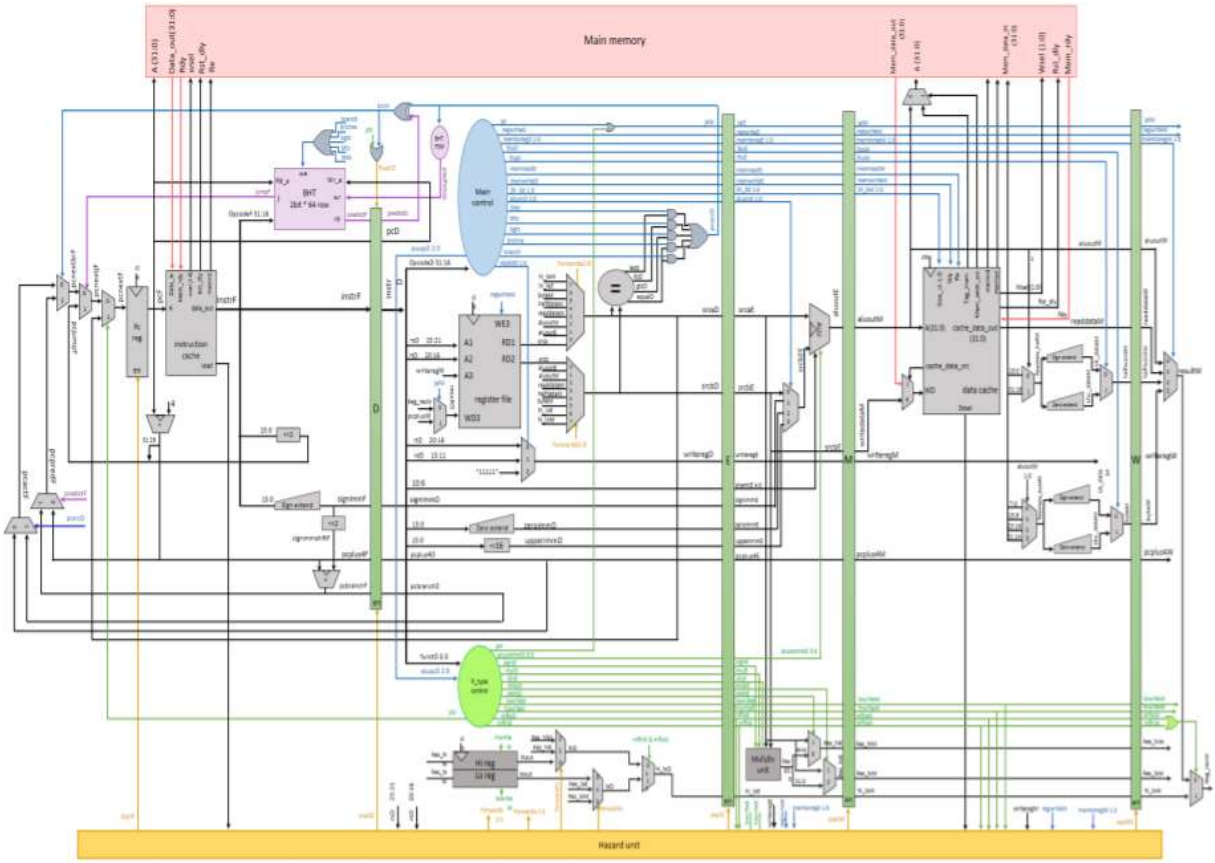

Fig 2 Architecture of proposed MIPS Processor

From this we have constructed the following blocks and simulated the results.

Instruction Cache

Figure 2.1, represents the architecture of the Instruction cache. The Instruction Cache can be enabled and disabled using the functions supplied. A function to invalidate the cache contents is also available. The valid bits in the Instruction Cache are cleared when it is invalidated, thereby emptying the cache of any loaded instructions. Instruction cache has a bigger impact on performance than data cache. This is due to the fact that the CPU is normally idle while an instruction is being fetched, whereas it can sometimes continue doing important work while a data fetch is in progress.

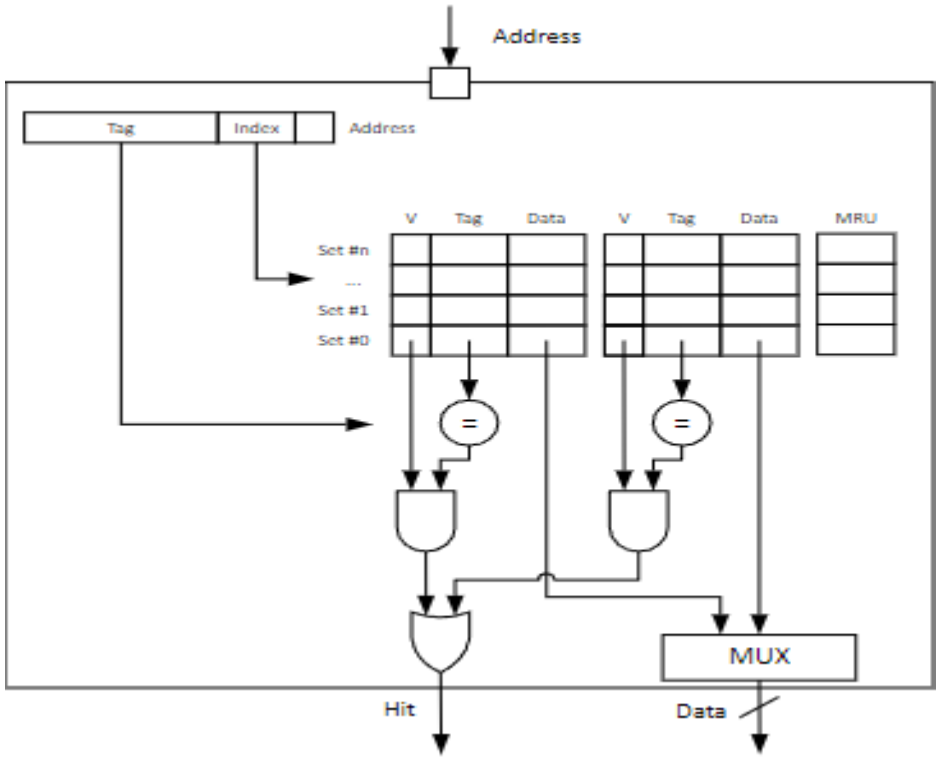

Figure 2.1 Architecture of the Instruction cache

Register File :

The register file is used by assembly language programmers to programme programmable registers.it's thought to be the hardware equivalent of a software array.The register file is made up of ports that allow you to read and write data in a specific index. In main memory and caches,there are similar interfaces for reading and writing data.The index of the register in a register file is represented by the register number. Memory address is used as an index in main memory.While a portion of memory address is utilised as an index in cache .As a result ,the register file is built up entirely of BRAM.

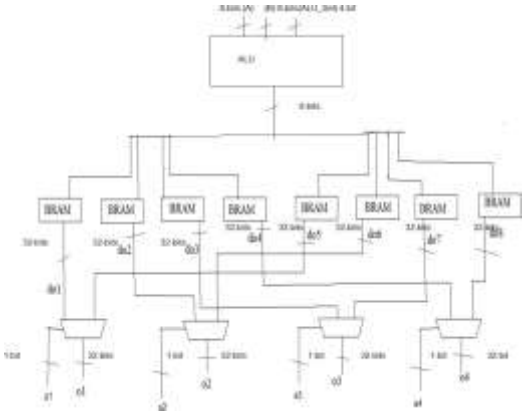

Fig 2.2 Architecture of register file

Arithmetic Logic Unit(ALU)

Figure 2.3,represents the architecture of ALU. The part of a central processing unit that performs arithmetic and logic operations on the operands of computer instruction words is called an arithmetic-logic unit. The ALU is split into two parts in some processors: an arithmetic unit (AU) and a logic unit(LU). Some processors have multiple AUs, such as one for fixed-point and another for floating-point operations.A floating-point

unit (FPU)on a separate chip called a numeric coprocessor is occasionally used in computer systems to do floating-point computations.

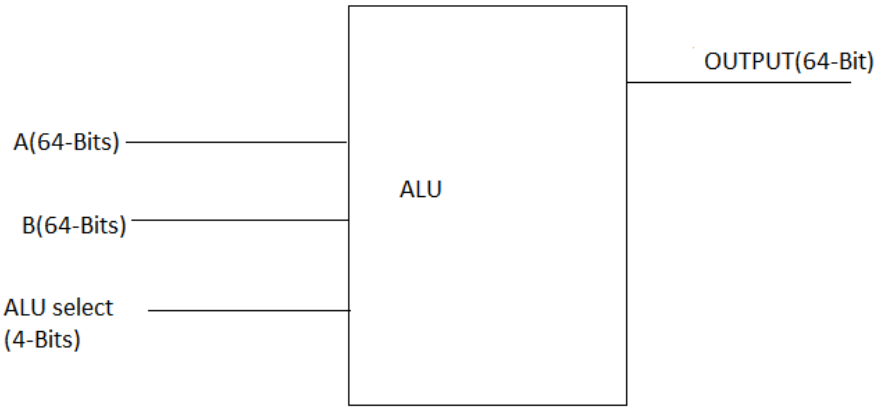

Figure 2.3 Architecture of Arithmetic Logic Unit

Main Control:

Decodes instructions to determine which segments of the data path will be active . Generates signals to determine which segments will be active in the data path. Muxes should be set to correct input. Decodes the ALU operation code .The register file can be read and written to (load/store) read and write to memory. Updates the program counter (branches).Computes the Branch Target Address.

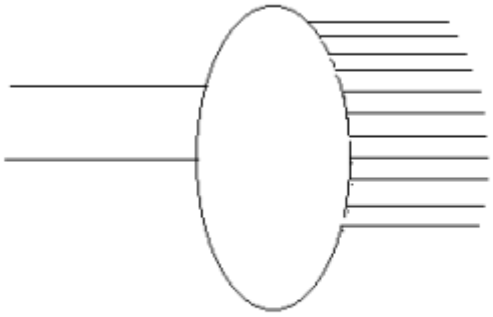

Fig 2.4. Main control

R-type control

When all of the data values used by an instruction are stored in registers, R instructions are used. The following is the format of all R-type instructions: rd, rs, rt OP.The mnemonic for the specific instruction is "OP." The source registers are rs and rt, and the destination register is rd. The add mnemonic, for example, can be written as: $s1, $s2, and $s3.

Branch History Table (BHT):

Branch prediction tackles problem of stalls from control dependencies. A Branch History Table keeps track of prior branch actions and targets, and predicts that future behavior will be similar. The outcome of earlier occurrences of branches (branching history) is utilized to forecast the present branch's result dynamically.

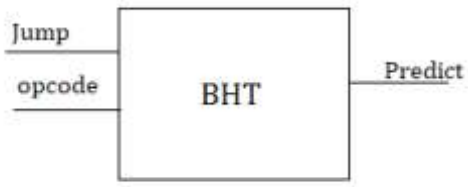

Fig 2.6. Branch History Table

Data Cache:

For writing a data in data cache, it's possible that a cache line doesn't hold the same data as the associated main memory block. “Write-through” and “write-back” rules are the two options for dealing with this problem. The write-through policy assures that the cache block and its main memory block are identical. Blocks can be copied from main memory to the data cache or vice versa for the data cache. We have a hit if the word is in the cache; the cache has the correct entry. To ensure that main memory and cache are consistent, the word is copied from the register to the cache and then returned to the appropriate block in main memory. If the desired block is not in the cache, an exception is thrown, referred to as a cache miss. The cache miss handler manages the transfer of the appropriate block from main memory to the cache.

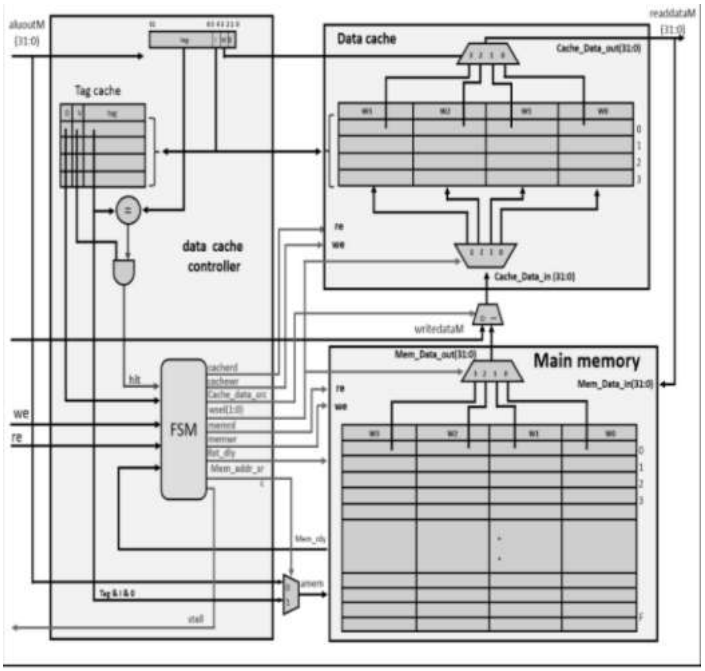

Fig 2.7. Architecture of Data Cache

Sign Extension :

The operation of expanding the number of bits in a binary number while keeping the number’s sign (positive/negative) and value is known as sign extension (abbreviated as sext). This is accomplished by appending digits to the number’s most significant side,according to a technique that varies depending on the signed number representation utilised.

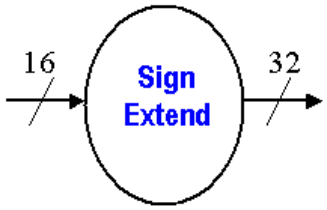

Fig 2.8 Architecture of sign extension

Zero Extend

Figure 2.9. shows the block diagram of zero extend.Zero extension is a similar idea (abbreviated as zext). Setting the high bits of the destination to zero, rather than a copy of the most significant bit of the source, is known as zero extension in a move or convert operation. If the source of the operation is an unsigned number, zero extension is usually the best way to move it to a larger field while keeping its numeric value, whereas sign extension is the best way to move signed numbers to a larger field while keeping their numeric value.

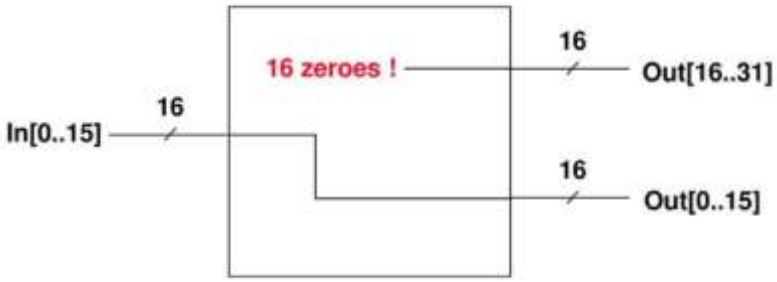

Figure 2.9 : Block diagram of Zero extend

PC Register File :

To give the ability to reset the incrementer to 0000 and latch the current address, the incrementer PC pin uses an S-R latch. The S-R latch utilised in the programme counter architecture has a gate level schematic illustrated below.

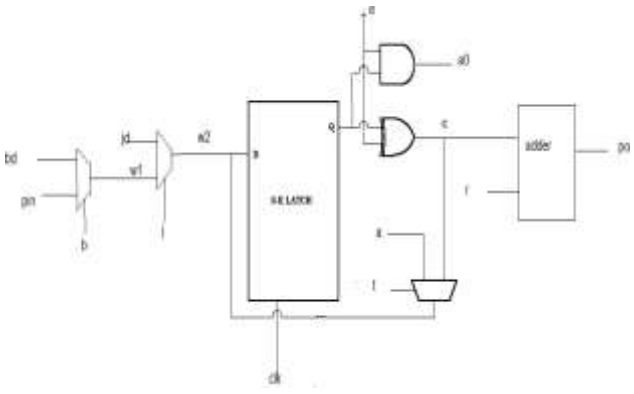

Fig 2.10 Architecture of pc register file

The address (position) of the instruction in a computer processor is stored in a register.

Hazard Unit:

When an instruction tries to read a register after a load instruction that writes the same register, forwarding will not assist. For the combination of load followed by an instruction that reads the result, something must halt the pipeline. As a result, in addition to a forwarding unit, a hazard detection unit is required. It works during the ID stage to put a stall between the load and its intended use.

Cache Controller:

A cache controller is in charge of the cache memory.The MIPS processor sends all addresses to the cache controller, which determines if the required data is in the cache. If it is, no memory access is required because the data is given straight from the cache to the MIPS processor; if it is not, the cache controller pulls many words from main memory in a row to fill the corresponding line in the cache. Each data cache and instruction cache has its own cache controller, which is made up of

1) Finite State Machine (FSM):

Instruction cache FSM varies from data cache FSM in that instructions are fetched by the CPU and executed without modification, whereas data is accessed for read or write.

2) Tag cache:

For each data cache line, the data tag cache has 26 tag bits, 1 valid bit, and 1 dirty bit. The 26 most significant bits of the address being read are stored in tag bits, the valid bit determines whether the cache line is valid, and the dirty bit is set when the cache line is overwritten without updating the related main memory block. When the machine reboots, all valid and dirty bits are reset. The instruction tag cache is identical to the data tag cache except that it does not contain dirty bits.

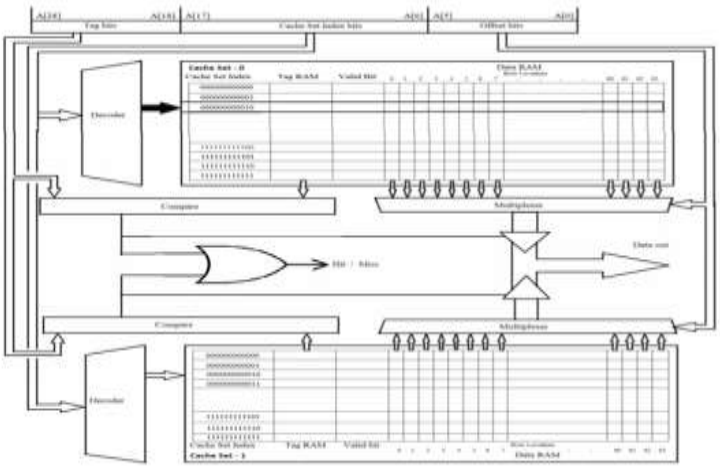

Fig 2.12. Cache Controller

Result and Discussion

In this paper, after completing the cache controller design, it is combined with a pipelined MIPS processor and programs were executed. And the proposed double precision unit will be used as one of the modules for designing an pipelined MIPS processor ,it was simulated and obtained results are listed below.

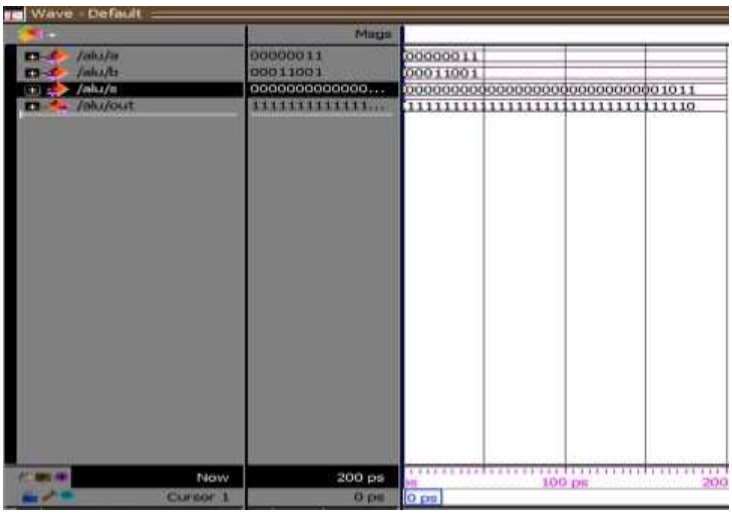

Fig 3.1 Simulation output of ALU

Figure 3.1 represents the simulation result of ALU which consists of two input, one output and select line for getting the opcode. Based on opcode ALU perform the operation and give outputs.

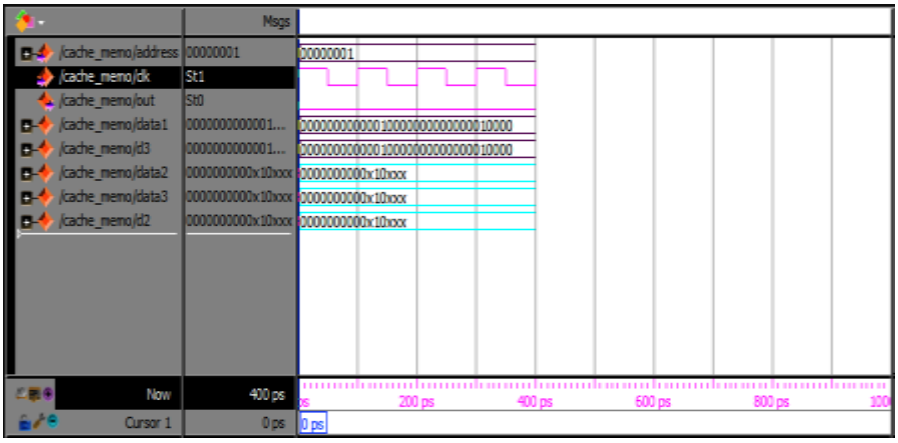

Fig 3.2 Simulation output of Direct mapping cache memory

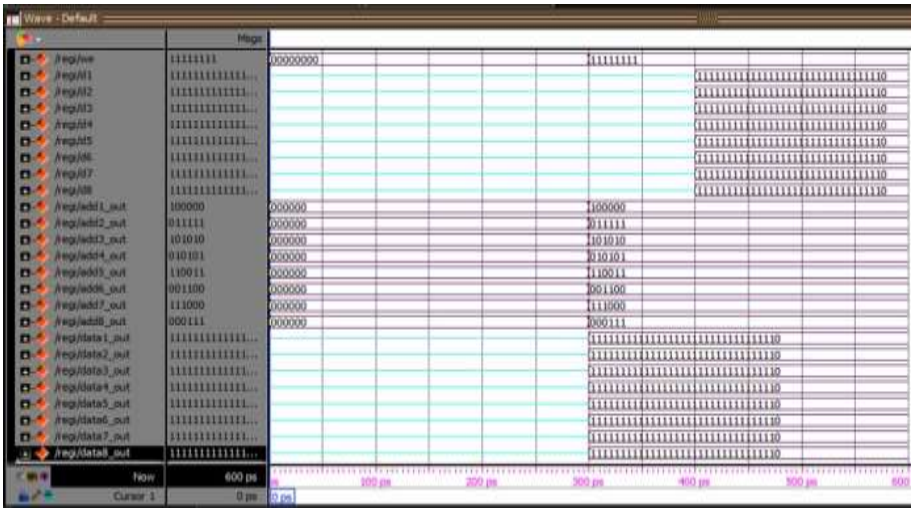

Fig 3.3 Simulation output of register file

Figure 3.3 represents the simulation output of 32 bit register bank. The register bank comprises of 8 BRAMs. All the 8 BRAMs have individual address which is given manually in which it store the same data

processed from the ALU. Now the data stored in BRAM is read and it if fed to 4 MUX. To reduce the number of data lines, MUX is used.

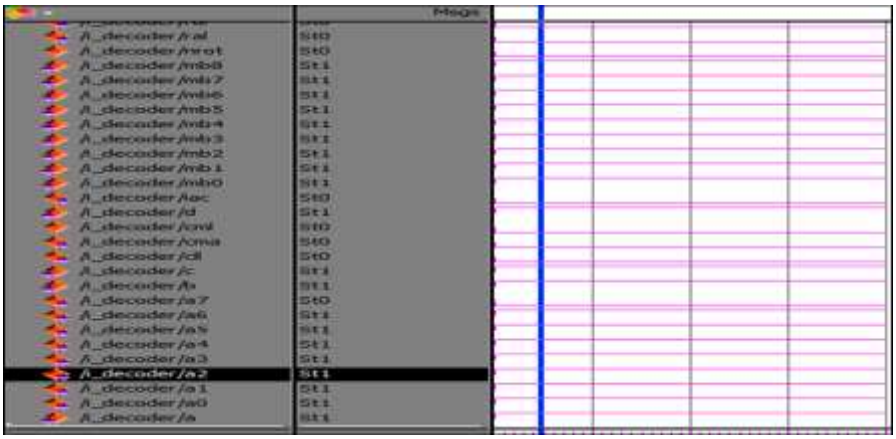

Fig 3.4 Simulation output of Instruction Decoder

Figure 3.4 represents the simulation output of Instruction decoder. Herethe output from instruction fetch is given as input to the decoder. After thedecoding the achieved instruction fed as input to the execution unit and Floating point unit with double precision.

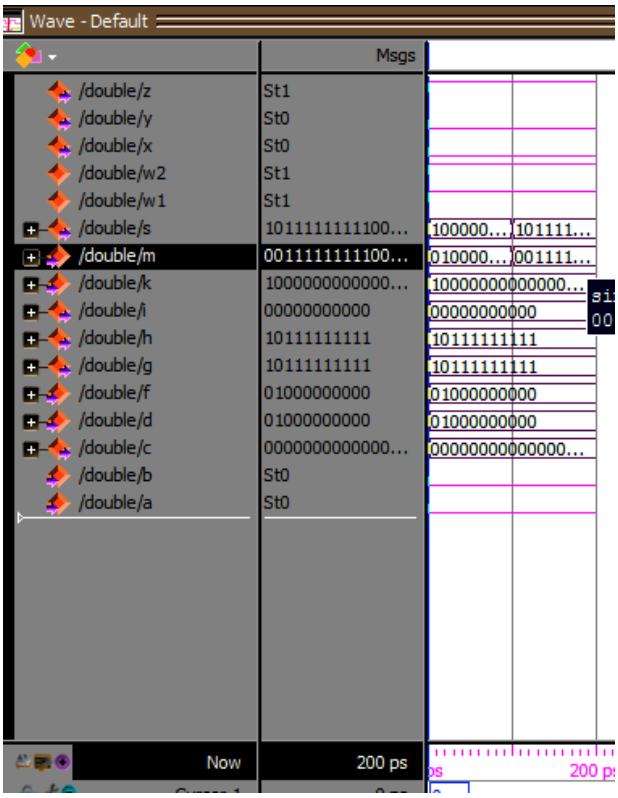

Fig 3.5 Simulation output of Floating point addition with double precision

Figure 3.5 shows the simulated output of Floating point addition with double precision a(sign), b(sign), e(exponent), f(exponent), m(mantissa), k(mantissa) are the inputs and s(sum) and c(carry) are the outputs.

Conclusion

In this paper a MIPS processor interfaced with a cache controller is designed and the execution is analysed. Compared to other processor as it has cache controller the speed of the processor is increased and by using

double precision the accuracy of the output is increased.

References

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).

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,2019- January.

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.

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, 2019- January.

Kaur, S., & Jassal, P. S. An Efficient Field Programmable Gate Array Implementation of Double Precision Floating Point Multiplier using VHDL.

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, 2020.

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.

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)

Acknowledgment

Deepika R ,currently pursuing B.E degree in Electronics and Communication Engineering from National Engineering College, Kovilpatti.

Gopika Priyadharsini S M**,** currently pursuing B.E degree in Electronics and Communication Engineering from National Engineering College, Kovilpatti.

Malini Praba M, currently pursuing B.E degree in Electronics and Communication Engineering from National Engineering College, Kovilpatti.

Muthu Malar M,currently pursuing B.E degree in Electronics and Communication Engineering from National Engineering College, Kovilpatti.

Vivek Anand. I received B.E degree in Electronics and Communication Engineering from Sethu Institute of Technology (2012) and M.E degree in VLSI from Mepco Schlenk Engineering College (2014). He is currently working in National Engineering College, Kovilpatti, India as Assistant Professor. His research interest includes Modeling and Simulation of Tunnel Field effect Transistors.