ASIC design of MIPS based RISC processor for high performance

Conference Paper · March 2017

DOI: 10.1109/ICNETS2.2017.8067945

2 authors, including:

Ravi V
Vellore Institute of Technology University
44 PUBLICATIONS **356** CITATIONS

ASIC Design of MIPS Based RISC Processor for High Performance

AGINETI ASHOK[1], V. RAVI[2*]
[1,2] School of Electronics Engineering, VIT University, Chennai, India
[1]aginetiashok@gmail.com, [2]ravi.v@vit.ac.in
* Address for correspondence

Abstract— **Objectives: The main aim of this paper is to implement 32Bit MIPS (Microprocessor Interlocked Pipeline Stages) RISC (Reduced Instruction Set Computer) Processor using Verilog HDL (hardware description language).**

Methods/Statistical analysis: The proposed algorithm analyzes the different stages of instruction decoding such as Instruction fetch module, Decoder module, Execution module and design theory based on 32Bit MIPS RISC Processor. In addition to that the algorithm uses pipelining concept which involves Instruction Fetch, Instruction Decode, Execution, Memory and Write Back modules of MIPS RISC processor based on 32Bit MIPS Instruction set in a single clock cycle.

Findings: RISC is a processor which is intended to perform a tiny set of operations, to expand the rate (speed) of the processor. In general, the processor works with a huge number of instructions every second by bringing the information from the memory. In the event that the processor speed does not coordinate with memory access speed then hardware interlocks happen. In concurring with this there is one more issue called stalls because of instruction pipelining in the CPU design. The primary desire of this paper is to design and synthesize the MIPS processor by making utilization of register files and to insert the ALU forwarding unit in order to avoid the stalls and hardware interlocks.

Application/Improvements: Based on the literature survey, the proposed method brings significant power efficiency improvements with enhanced performance and reduced power dissipation due to not only technology scaling but also a great deal of design efforts.

Keywords: Inter locked pipeline stages; Hazards; Register files; ALU forwarding unit.

1. INTRODUCTION

The MIPS instruction set architecture is a RISC based chip design. MIPS processors are mainly used in Cisco Routers, digital cameras, Sony play station game consoles and Windows CE devices. MIPS contain fixed length straight forward decode instruction format, where the load and store has the constrained memory, large number of register record where the operations can have finished inside of the register record of the processor. The primary issue with MIPS processor is Data Hazards and is characterized as a circumstance where the pipelined processor creates an erroneous yield because of information reliance relations between the instructions. This happens when the information of one instruction concurs with the yield of a past instruction, yet the past instruction output won't be present at the time of executing the present instruction. At the point when the instruction execution stage happens then the both instructions have information reliance. This can be avoided by using nop's (no operation) or the result of the ALU is forwarded until the output of the previous instruction is stored in the register file. In a single clock cycle of MIPS single cycle processor, it carries only one instruction. It performs the tasks like fetching the instruction, decoding the instruction, memory access, executing the instruction and writing back all the results in single clock cycle. In every single cycle of an MIPS processor, register record supports two autonomous register read and single register writing back. The register record reads in the memory locations with the help of the requested address and yields the information values contained in the register. On this information ALU can be worked, whose operation is dictated by the control unit to

to either process a memory address, perform arithmetic operation (sub and addition), or performing compare (branch). If the instruction decoded is arithmetic, then the result from ALU must be written back to a register. If the instruction decoded is a store or load, then the ALU result can be utilized as a address to the data memory. The last step composes the ALU result or memory esteem back to the register record.

2. RISC AND CISC ARCHITECTURE

2.1 CISC Architecture

CISC architecture is essentially the chips which are effectively programmable; additionally, improve proficient and memory utilization. It was produced to streamline the compiler advancement. For instance, a CISC processor has an inherent ability to execute complex instructions, rather than making compiler compose long machine instructions. Pentium is an illustration of CISC processor. The given assignment is executed by utilizing the less number of instructions. Slow memory is utilized to make it more proficient.

Compiler is less muddled as smaller scale programming instructions which are written to match high level constructs. For each new era of directions for a PC have a subset of past form set of instructions[1]. Subsequently it turns out to be more unpredictable with each most recent era PC's. As the instruction is of any length, subsequently every instruction will take diverse number of clock cycles to execute. Consequently, the general execution is slowed down.

2.2 RISC Architecture

It utilizes little and very advanced arrangement of instructions. Each and every instruction uses a solitary clock cycle to execute. RISC makes utilization of pipelining idea where it permits the concurrent execution of instructions, by making the processor more effective. To stay away from the gigantic number of cooperation's with memory, RISC utilizes an extensive number of register record to store the intermediate results/data.

3. Five Stage Pipelining

The execution of an instruction in a processor is spitted into number of stages, but in the design of MIPS processor I make use of five stage pipelining. Figure.1 shows five stage pipelining.

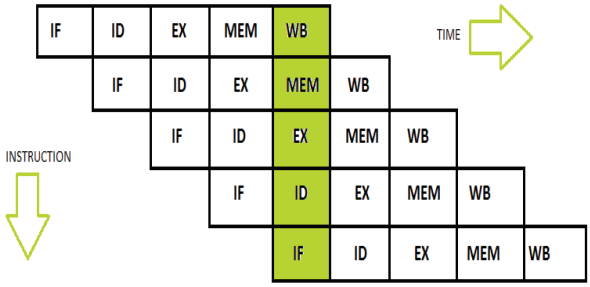

Figure 1. Five Stage Pipelining

The IF stage gets the next instruction from memory with the address present in the Program Counter (PC) and then it will be stored in the instruction register (IR). Whereas in ID stage decodes the instruction present in the instruction register and evaluates the program counter instruction, and reads if any operand is needed from register records[2].

In EXE stage the execution of instructions takes place, where all arithmetic and logical operations such as shifts left and right, subtraction, addition are done in this stage. If any current instruction requires the memory access, then the Memory Access stage will perform this type of operations. So, for load instruction, Memory Access stage would load an operand from memory.

4. Proposed Work

4.1 Hazard Unit

Mainly the hazards occur due to instruction pipelining in central processing unit. An incorrect compilation result occurs when the next instruction cannot execute in the respective clock cycle. The control logic determines whether a hazard will occur or not when an instruction is fetched. If the hazard is going to happen then the control unit will insert no operation.

4.2 ALU Forwarding Unit

As the processor executes millions of instructions per second, then there is a problem called Interlocking in the pipeline stages and the solution for it is to stall the pipeline stages. Divide the pipeline into two parts, instruction fetch and instruction execute as shown in Table 1.

Table 1. Division of pipeline stages.

IF Phase		EXE Phase		
IF	ID	EXE	MA	WB

So, to stall the pipeline ALU forwarding unit is used in order to make use of the ALU result directly, without waiting for the result to be written back to the register record. In order to get the result like that we need to forward the result of the ALU directly back to the arithmetic and logical unit. The pipeline register record contains the ALU output produced by an ALU. So, this result has been forwarded to the subsequent instructions in order to prevent the data hazards. It selects the correct ALU inputs to the execution stage. If the hazard occurs, the operands will come from either MEM/WB or EX/MEM pipeline registers. This is shown in Figure.2

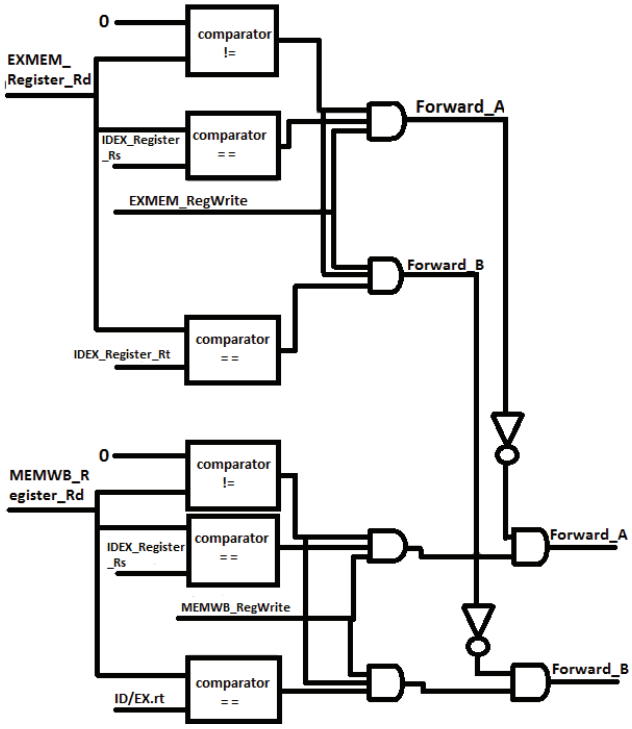

Figure 2. ALU forwarding unit

If there are no hazards, then register file will provide the operands for an ALU[3]. The arithmetic and logical unit sources can be chosen with the help of two multiplexers, with control signals namely Forward_B and Forward_A shown in Figure.3. So, the forwarding Unit removes the data hazards which involves in arithmetic instructions. It just compares the source register of the current instruction and the destination register record of the previous instruction.

4.3 Instruction Divider

As the MIPS processor is 32 bit it makes use of 32-bit general purpose registers. MIPS has three types of instructions they are

1. I-type: This type of instruction format is used Load and Store instructions.
2. R-type: This type of instruction format is used for Arithmetic Instructions.
3. J-type: This type of instruction format is used for jump instructions.

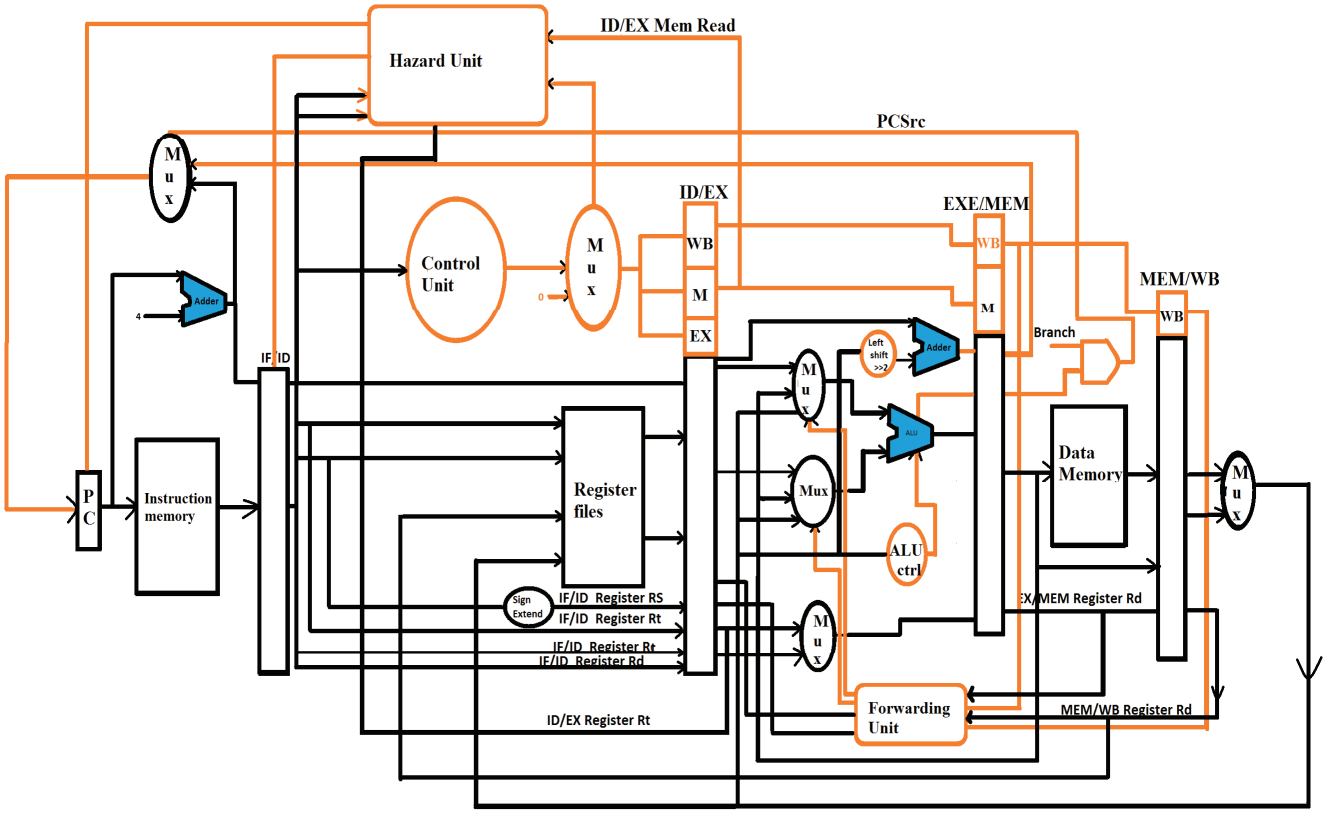

Figure 3. MIPS processor with forwarding unit

Field	Description
opcode	6-bit primary operation code
Rd	5-bit specified for the destination register
Rs	5 - bit specify for the source register
rt	5 - bit specified for the Target(rs/rd) registers or used to specify function within the primary opcode
Address / immediate	16 – Bits signed immediate used for logical operands, arithmetic signed operands, load/store address byte offsets, and PC- relative branch signed instruction displacement
Inst index	26 – bit index shifted left two bits to supply the low-order 28 bits of the jump target address
sa	5 – bit shift amount
function	6 – bit function field used to specify function within the primary code

Figure 4. Different Fields of instruction format and its description

In Figure 4, the three instruction format uses six-bit opcode to perform particular operations like addition, subtraction depending on the opcode value. I type instructions includes branch Instruction.

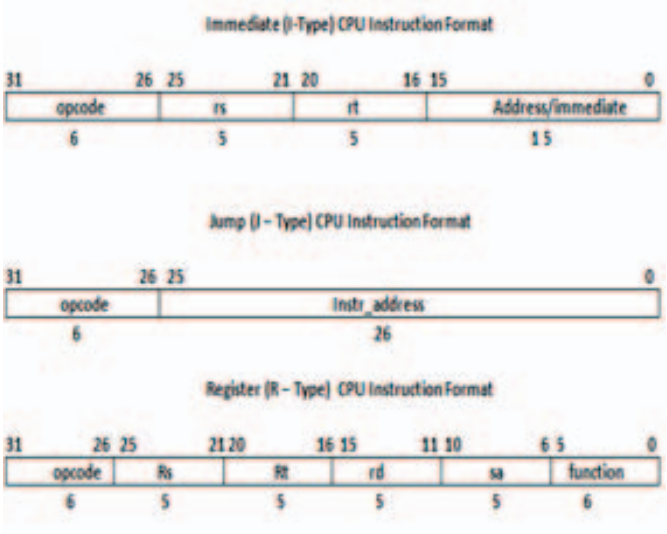

Figure 5. Different types of instruction format

To calculate the branch destination address, branch instruction uses the sum of offset value from the present address in the address in accordance with the program counter. Source register (Rs) and destination register (Rd) are of 5 bits to store the offset values. Where J type instruction uses 6-bit opcode and 26 bits of address, where the two least significant bits are removed and the four most significant bits are removed to get as same as the current instruction address[4]. R type instruction is used for performing ALU operation based on the opcode. The register is used to store results and shift amount (sa) is used to shift and rotate instructions. The amount of shift is decided by the source operand Rs is shifted. Function is used because it contains the control codes to differentiate the multiple instructions[5]. The different fields in the instruction divider with its description are as given in Figure.5.

5. Simulation Results

5.1 Execution Unit

MIPS execution unit contains ALU, which perform the operations based on the opcode. By the addition of program counter value to the sign extension unit, which is left shifted by two units with the help of an adder we will get the branch address The sign extended unit increases the number by appending the most significant bit in order to preserve the sign of a binary number[6]. The control signals for ALU are generated by the ALU controller. ALU controller is a circuit has two inputs followed by and output, which is a two-bit data that tells ALU, which type of arithmetic and logical operation that ALU performs on the two input data[7]. The simulation results are carried out in a tool called Xilinx and the resultant waveform for execution unit is shown in Figure 6.

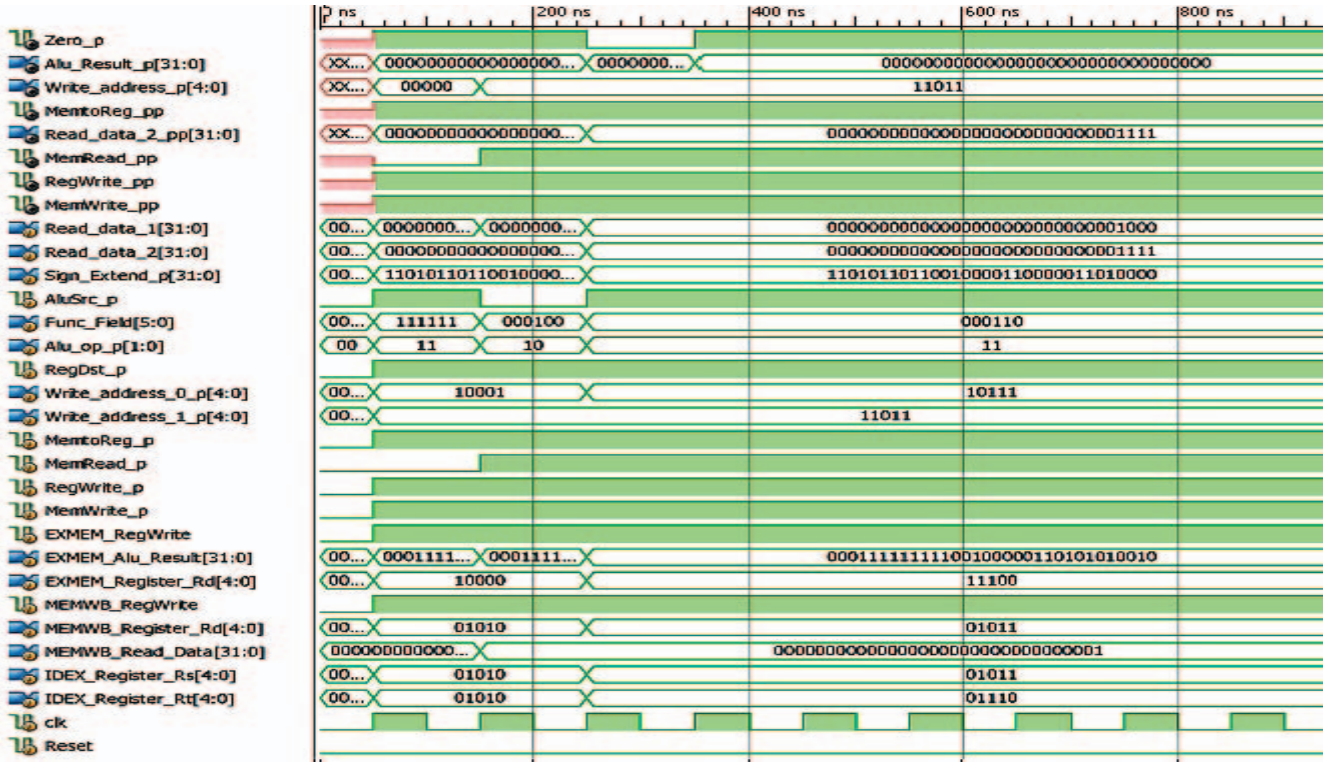

Figure 6. Execution unit output waveform

5.2 Hazard Unit

This Hazard detection unit detects whether there is a stall in the pipeline or not. Figure 7 shows the simulation result for hazard unit. If the signal IDEX_MemRead is active high and IDEX_Register_Rt is equals to IFID_Register_Rs or IDEX_Register_Rt equal to IFID_Register_Rt then stall is active high otherwise stall will be active low.

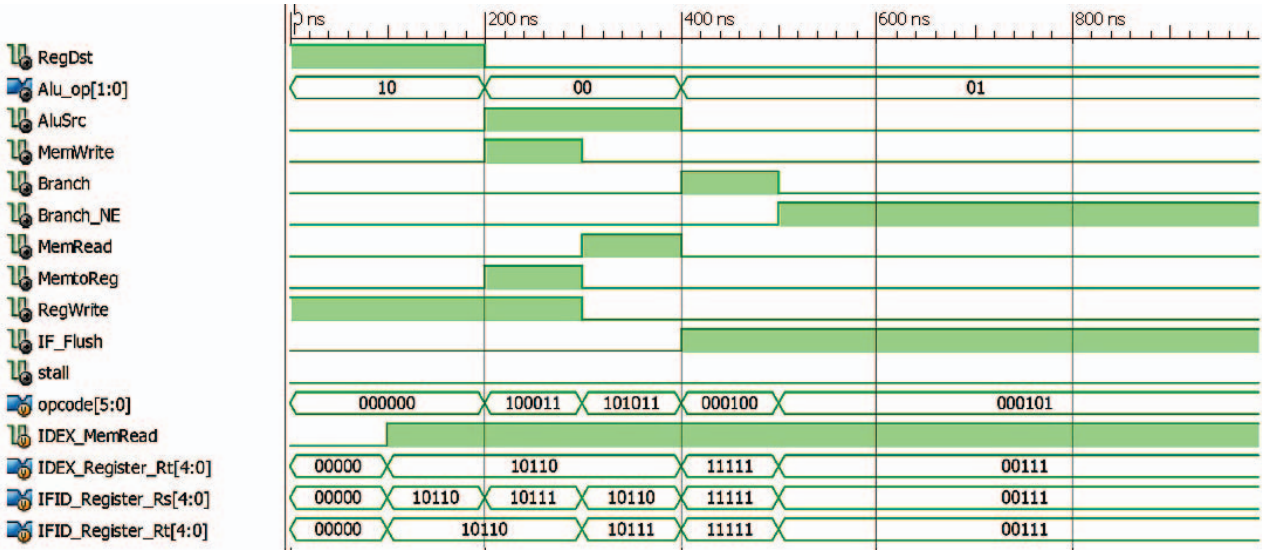

Figure 7. Hazard detection unit waveform

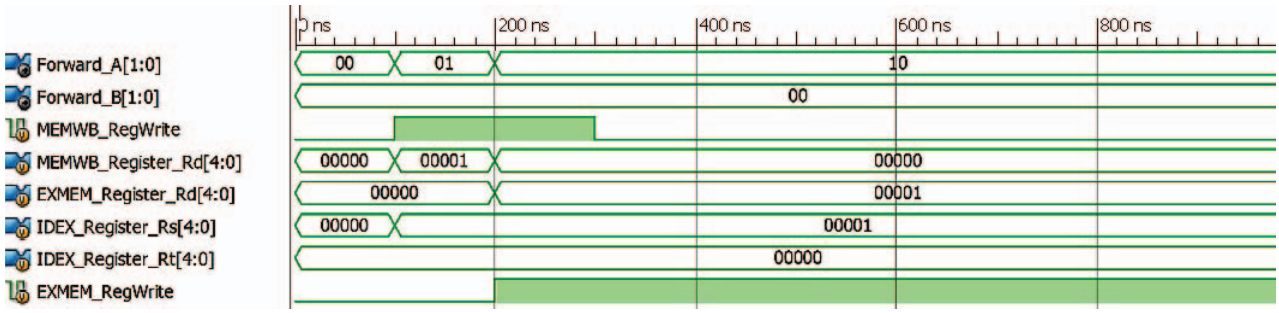

Figure 8. ALU forwarding unit waveform

5.3 ALU Forwarding Unit

It consists of comparator which compares the singles and produces the output. Figure 8 shows the simulation result for
ALU forwarding unit and the outputs are Forwar_A and Forward_B.

6. Layout

Layout for the processor shown in Figure 9 is done using cadence tool called SoC Encounter. The inputs for the tool are gate level netlist, standard cell library files, timing constraints file and technology file[8]. Gate level netlist is consisting of interconnected logic gates that define the logic. Standard cell library has readily made logic cells like nor gate, and gate etc. A netlist has the instantiation of these standard cells. Technology file has the rules for the design like metal widths, spacing etc and the cadence tool will take .lef file as technology file. While doing synthesis for the design the timing constraint file is generated, which contains the timing constraints of the design like input and output delay constraints, false paths and clock definitions[9]. The major step in the layout is floorplanning where the chip quality is determined. In this step the space for standard cells, macros in the design, routing resources for power and size of the chip is defined. Aspect ratio defines the height and width of the chip. Then the core of the chip is utilized up to 70% to place standard cells and macros[10]. The remaining 30% is used for routing the gates and if we want to place the buffers. Core utilization formula is given as.

Core Utilization = (Standard cell area + macro cell area) / total core area

After Floor planning the power rings and strips were added followed by routing and timing verification for the design. The final netlist is compared with the desired netlist.

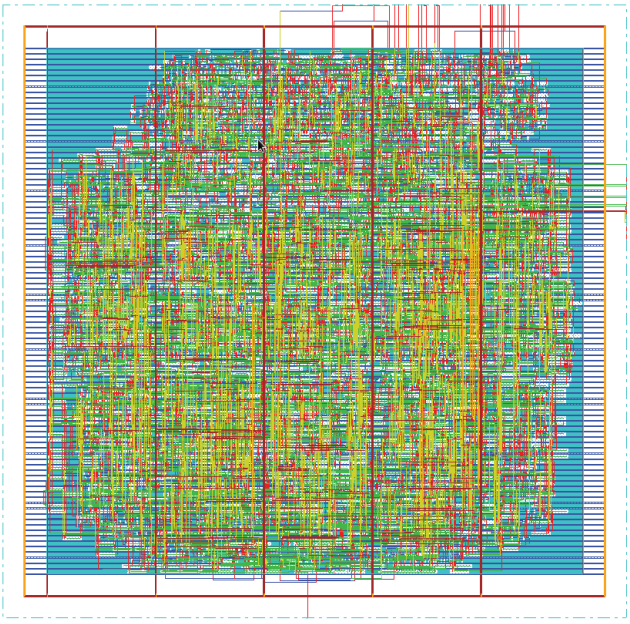

Figure 9. ALU forwarding unit waveform

Power calculation is tabulated in terms of the cells used for each phase of an instruction and the leakage power, dynamic power; total power is shown in the Table 2.

Table 2: Power calculations

Modules	Cells	Leakage Power (nW)	Dynamic Power (nW)	Total Power (nW)
Processor	4.231	3667.04	17421614.78	17425281.82
IF Unit	190	133.041	437604.073	437737.144
ID Unit	2788	2777.06	13636676.24	13639453.3
EXE Unit	1119	566.164	1097609.007	1098175.171
Data Memory	74	130.795	794861.522	794992.317
Control Unit	30	18.171	8060.935	8079.105

7. Conclusion

MIPS process is a best contender to remove the hazards in original datapath with the help of forwarding unit, where by fetching the results from the pipeline registers before they return back to the register record. So, the processor won't go to the high impedance or unknown state, which intern results in the performance enhancement.

References

1. W. Hu et al., "Godson-3B1500: A 32 nm 1.35 GHz 40 W 172.8GFLOPS 8-Core processor," in IEEE ISSCC Dig. Tech. Papers, 2013, pp. 54–55.
2. W. Hu et al., "Godson-3B: A 1 GHz 40 W 8-Core 128GFLOPS processor in 65 nm CMOS," in IEEE ISSCC Dig. Tech. Papers, 2011, pp. 76–78.
3. Sinha, Neha, and V. Ravi. "Implementation of health monitoring system using mixed environment."
4. Indian journal of science and technology
5. 8.20 (2015): 1.
6. B. Fan et al., "Physical implementation of the 1 GHz Godson-3 Quad-Core microprocessor," J. Comput. Sci. Techn., vol. 25, no. 2, pp. 192–199, 2010.
7. W. Hu and Y. Chen, "GS464V: A high-performance low-power XPU with 512-bit vector extension," in Hot Chips Symp., 2010.
8. J. Friedrich et al., "Design methodology for the IBM POWER7 microprocessor," IBM J. Research and Development, vol. 55, no. 3, pp. 9:1–9:14, 2011.
9. M.Dodiya Chandni. M and V.Ravi, "Built in Self-Test Architecture using Concurrent Approach", Indian Journal of Science and Technology, Vol 9(20), May 2016
10. Q. Fan et al., "A synchronized variable frequency clock scheme in chip multiprocessors," in Proc. IEEE ISCAS, 2008.
11. S. Damaraju et al., "A 22 nm IA multi-CPU and GPU system-on-chip," in IEEE ISSCC Dig. Tech. Papers, 2012, pp. 56–57
12. Charles E. Gimarc, Veljko M. Mhtinovic, "RISC Principles, Architecture, and Design", Computer Science Press Inc., 1989.