ISBN 978-981-11-3671-9
Proceedings of 2017 the 7th International Workshop on Computer Science and Engineering (WCSE 2017)
Beijing, 25-27 June, 2017, pp. 405-409
doi: 10.18178/wcse.2017.06.069

Implementation and Performance Evaluation of Pipelining Mechanism in 32-bit MIPS Architecture

Nan Wang[+], Ranjith Kumar and Rajath M. Basavaraj

California State University, Fresno

Abstract. The demand for increasing the speed of instruction execution in processors and instruction accessing from memory leads to instruction level parallelism and the design of memory structures closer to the central processing unit. Such esteemed and efficient processors are built only after forming the base in the VLSI front end design stages. A microprocessor without Interlocked Pipeline Stages is a 32-bit RISC instruction set architecture. The principle used in MIPS design is to form pipeline stages such as fetch, decode, execute, memory access, and write back and make them work together efficiently in an overlapped way. The critical instruction types like Register type instructions, Load and Store instructions, and Branch instructions are taken into picture by considering the instruction set of a known processor. This paper aims at implementation and simulation of the 32-bit MIPS architecture as a RISC processor. The completed 32-bit MIPS architecture with pipelining mechanism is designed and simulated to produce outputs including execution time and propagation delays. The results are then analyzed and compared to show efficiency of the pipelining scheme by calculating CPIs with and without data dependency between the input instructions.

Keywords: 32-bit MIPS design, pipelining, verilog HDL, RISC, CPI.

1. Introduction

The Central Processing Unit, also referred as the CPU is the hardware design inside a computer system which performs based on the instructions given by a computer program. There are two main types of popular processors, RISC (Reduced Instruction Set Computing) and CISC (Complex Instruction Set Computer) processors. CISC has a large amount of different multi-clock complex instructions. This type of processor is slower in comparison to RISC; but at the same time it uses fewer instructions. The RISC processor uses simpler and faster instructions that are typically of one size. Therefore, theoretically it uses fewer transistors which make RISC processors easier and less expensive to design.

A MIPS (Microprocessor without Interlocked Pipeline Stages) architecture is designed to employ three different types of instructions which are R-type (Register type), I-type (Immediate type) and J-type (Jump type). Those instructions which carry out arithmetic or logical operations between two registers are categorized into R-type instructions. I-type instructions deal with the operations between a register and an immediate number to generate memory address for load or store instruction and arithmetic result for immediate operations. Conditional branch instructions are also I-type instruction. J-type instruction takes a pseudo target address and will branch unconditionally.

This paper describes the front end implementation and verification of pipelining mechanism in a 32-bit MIPS architecture. Performance comparison between different MIPS designs are also presented based on the simulation results. The organization of this paper is as follows. Section 2 discusses the concept of pipelining technique. Section 3 presents the MIPS design procedures. Design implementation, verification and simulation results are presented in section 4. Finally, Section 5 concludes the paper.

[+] Corresponding author. Tel.: + (001) 304-610-7731; fax: +(001) 559-278-6297
E-mail address: nwang@csufresno.edu

2. Concept of Pipelining

Pipelining is an implementation technique whereby multiple instructions are overlapped in execution; it takes advantage of parallelism that exists among the steps needed to execute an instruction. All recent processors incorporate pipelining as a key implementation technique. MIPS is a five stage pipelining structure; each stage is responsible for completing a part of an instruction as explained above. These five stages are connected through the pipelining registers as shown in Fig. 1. The throughput of the pipeline is determined by how much time an instruction takes to be executed. The time required to move an instruction one step down to another stage among five stages sequentially is known as 'processor cycle'.

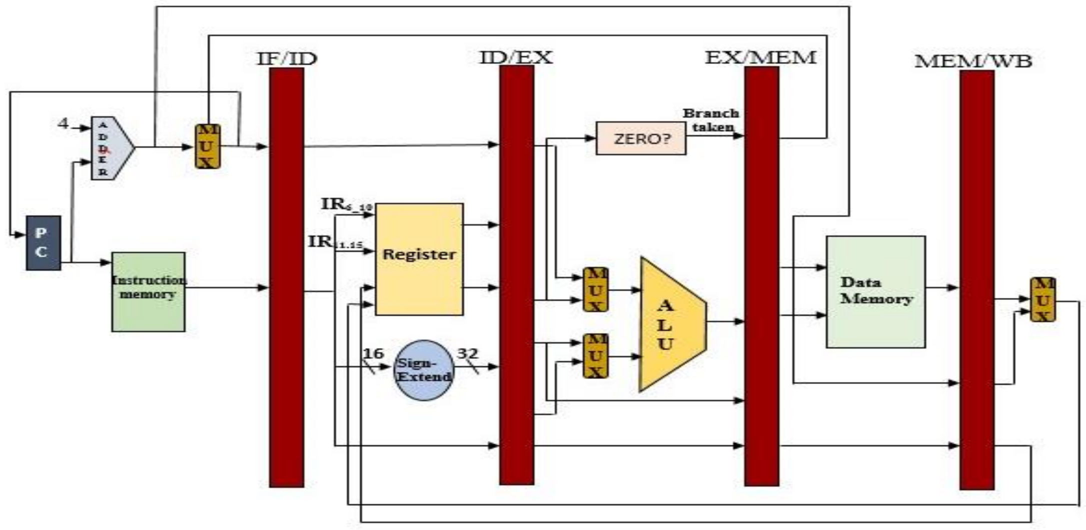


Fig. 1: MIPS block diagram [1].

The slowest pipeline stage decides the length of the processor cycle. It is the designer's responsibility to balance the length of processor cycle of each stage. Let us consider that stages are balanced then the time for each instruction on processor can be determined by the result of Time per Instruction on non-pipelined Machine divided by the Number of Pipeline Stages [1]. Each clock cycle means one of the stages in the pipeline. Even though an instruction takes five clock cycles to complete, the CPU will start a new instruction and will execute a step of the instruction at each of the following clock cycles. For example, the processor executes a stage form all the instructions at clock cycle number 5.

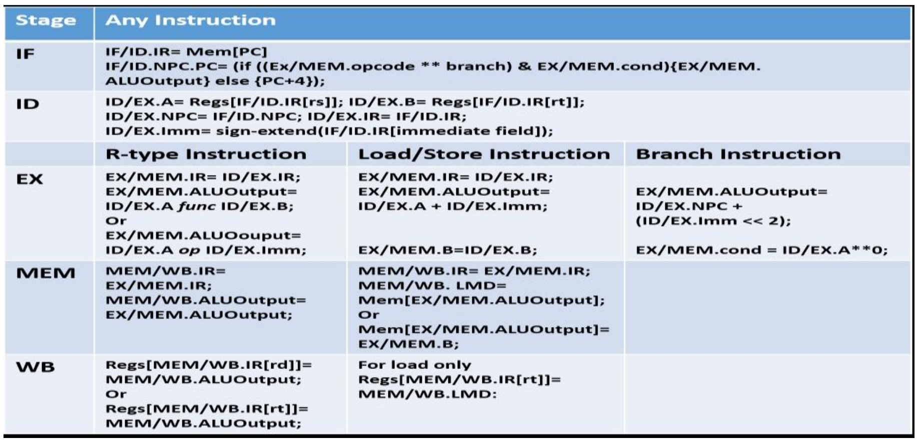
3. Implementation Procedure

3.1. MIPs instruction format

In MIPS there are 3 different types of instructions, R-Type (Register Type), I-Type (Immediate Type), J-Type (Jump Type). R-type instructions take 3 different arguments: RT and RS are source registers and RD is the destination register. I-type instructions take two arguments, RS, RT and a 16-bit immediate value. J-type instructions are written with labels and it is the linker or assembler's duty to convert the label into numerical target address.

3.2. Pipelining operations

A new instruction is fetched from instruction memory and updated PC (Program Counter) is stored into both of the pipeline register as well as the PC register. The register values are fetched from register file; the 16-bits of immediate value is sign extended; the opcode and function section are used to create control signals. All ALU operations are performed in the ALU. It can be arithmetic or logic operations for R-type instructions, addition to calculate the memory address for Load/Store instructions and comparison for Branch instructions. Data is written to memory for Store instructions and read from memory for Load instructions; new program counter value can be generated here. Data is written back to register file from ALU output for R-type instructions or from memory for Load instructions. The operations to be performed in each stage of pipeline are illustrated in Fig. 2.



Stage	Any Instruction		

IF	IF/ID.IR= Mem[PC]
	IF/ID.NPC.PC= (if ((Ex/MEM.opcode ** branch) & EX/MEM.cond){EX/MEM. ALUOutput} else {PC+4});		

ID	ID/EX.A= Regs[IF/ID.IR[rs]]; ID/EX.B= Regs[IF/ID.IR[rt]];
	ID/EX.NPC= IF/ID.NPC; ID/EX.IR= IF/ID.IR;
	ID/EX.Imm= sign-extend(IF/ID.IR[immediate field]);		

	R-type Instruction	Load/Store Instruction	Branch Instruction

EX	EX/MEM.IR= ID/EX.IR;
	EX/MEM.ALUOutput=
	ID/EX.A *func* ID/EX.B;
	Or
	EX/MEM.ALUOouput=
	ID/EX.A *op* ID/EX.Imm;
		EX/MEM.IR= ID/EX.IR;
		EX/MEM.ALUOutput=
		ID/EX.A + ID/EX.Imm;
		
		EX/MEM.B=ID/EX.B;
			EX/MEM.ALUOutput=
			ID/EX.NPC +
			(ID/EX.Imm << 2);
			
			EX/MEM.cond = ID/EX.A**0;

MEM	MEM/WB.IR=
	EX/MEM.IR;
	MEM/WB.ALUOutput=
	EX/MEM.ALUOutput;
		MEM/WB.IR= EX/MEM.IR;
		MEM/WB. LMD=
		Mem[EX/MEM.ALUOutput];
		Or
		Mem[EX/MEM.ALUOutput]=
		EX/MEM.B;	

WB	Regs[MEM/WB.IR[rd]]= MEM/WB.ALUOutput; Or Regs[MEM/WB.IR[rt]]= MEM/WB.ALUOutput;	For load only Regs[MEM/WB.IR[rt]]= MEM/WB.LMD:	

Fig. 2: The operations in each stage of pipeline in MIPS architecture [1].

3.3. Pseudo LRU for cache line eviction

For every memory operation in the memory access stage, the processor has to search the address location down the data memory bank. A cache structure can be implemented between the CPU and the data memory. Pseudo LRU (least recently used) is a Cache eviction algorithm where a cache line is evicted when the cache is completely filled. It is similar to True LRU, but uses fewer bits. The Pseudo LRU uses a tree like structure where the direction of each bit gives the least recently called bit as shown in Fig. 3.

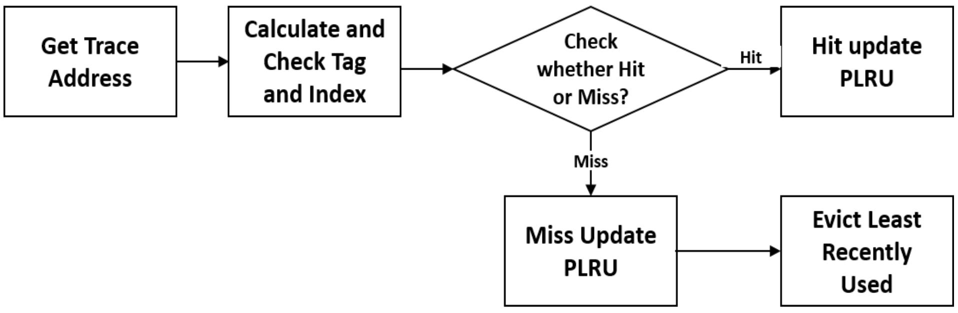

Get Trace Address
Calculate and Check Tag and Index
Check whether Hit or Miss?
Hit
Hit update PLRU
Miss
Miss Update PLRU
Evict Least Recently Used

Fig. 3: Operations of the pseudo LRU

The Pseudo LRU algorithm can adapt itself to multiple ways which can be initialized at compilation time [8]. The steps can be depicted below.

Firstly, each index of the cache structure has a 2 dimensional array to store the Tree bits for that particular index; Secondly, two separate tasks are used for updating the LRU bits in case of a hit or a miss. Thirdly, in case of a hit, the task simply sets or resets the bits in the tree corresponding to the way at which the hit occurred; Fourthly, in case of a miss, the task traverses through the tree and finds its path by reading the bit values at the nodes of the tree and reaches the particular way; Lastly, after that way has been evicted, hit update LRU task is called to ensure that the previous line which was written into is not accessed any time before the other lines in the index are accessed.

4. Implementation and Simulation Results

A 32-bit binary input was given to the MIPS design through a MIF file with instruction input. All simulations are carried out on ModelSim simulation environment by Mentor Graphics. ModelSim is a multi-language HDL (Hardware Description Language) simulator.

4.1. MIPS pipelining architecture simulation

In this section, the above verified instructions are given as inputs to the pipelined MIPS architecture. To verify the efficiency of the pipelining mechanism, simulations are carried out on instruction inputs with/without data dependency. Data dependency happens when the operations to be performed on the consecutive instructions depend on the results of the previous instruction.

4.1.1 Pipelining without data dependencies

The seven input instructions to the design are, Load, Load, Add, Store, Ori, Jump, and Beq instructions without any data dependency between them. The simulation waveform is shown in Fig.4. Each instruction used is denoted by a sequence number and a same color is used for highlighting and representing the operation results processed in each stage for that instruction as shown in Figure 6. Each instruction execution result shown in Figure 6 starts with highlighted current PC value. The pipelining can be clearly observed looking at the overlapping of the instructions executions during the operations of one instruction, there are operations also being performed on the other instructions at a particular clock cycle. The sequence of instructions is executed and the total execution time is observed as 27000 Pico seconds.

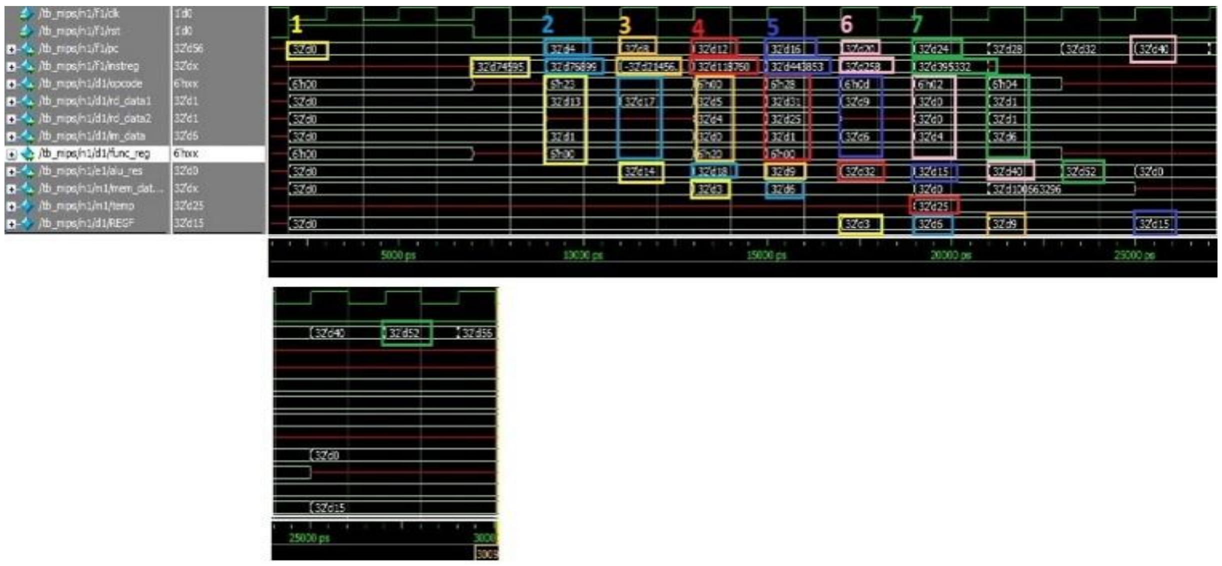


Fig. 4: Waveform of simulation result not having data dependencies

4.1.2 Pipelining with data dependencies

The input instructions are, Load, Load, Add, Store, Ori, jump, Beq instructions with data dependencies between them. The simulation waveform is shown in Fig. 5. The pipelining execution in the waveform can be observed and clearly understood when followed by the same color pattern highlighted for each instruction execution operation. Same color pattern highlighting has been used and the notations of each stage is represented by the instruction sequence number.

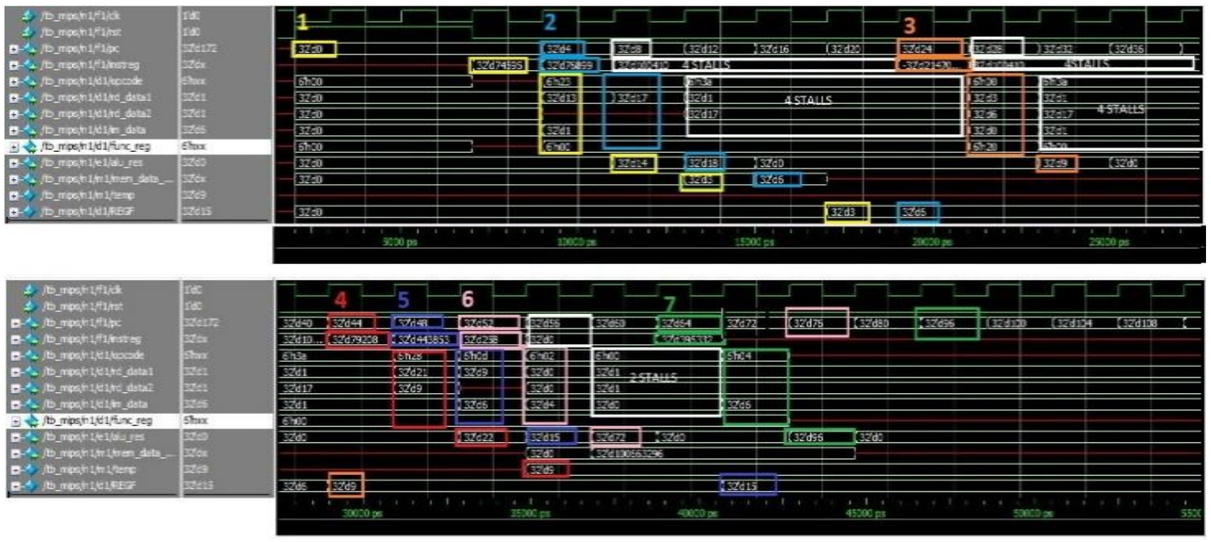


Fig. 5: Waveform of simulation result with data dependencies

There are 4 stalls installed between the second and third and the third and fourth instructions to overcome the data dependencies between the instructions because the following instruction can be executed only when the result of the last instruction is available. There are two stalls installed between sixth and the seventh instruction. The total execution time is observed as 46500 ps.

4.2. Simulation results and comparison

The simulation results and comparison between different MIPS implementations are shown in Table 1.

Table 1: Simulation results and comparison

Implementation	Total Execution time	Total Clock Cycles	Clock Cycle Time	CPI
Non-pipelined MIPS	77 ns	7	11 ns	1
Pipelining with Dependencies	46.5 ns	23.25	2ns	3.32
Pipelining W/O Dependency	27 ns	13.5	2ns	1.93

Based on the simulation results, some justifications can be drawn.

- For the non-pipelined MIPS implementation, the clock cycle time has to be long enough for the worst case instruction which is Load instruction at 11 ns (5.5*2 ns). Therefore, the total execution time of 7 instructions is 77 ns.
- For the pipelined MIPS implementation with data dependencies shown in Figure 7, a total execution time of 46500 ps or 46.5 ns is observed with a clock cycle time of 2 ns. Total number of clock cycles can be calculated as 46.5 / 2=23.25 clock cycles which suggests the value of CPI (Clock Cycles Per Instruction) =23.25 / 7 ≈3.32. It runs 65.6% (77/46.5 ≈1.656) faster than the non-pipelined MIPS implementation in terms of execution time.
- For the pipelined MIPS implementation without data dependency shown in Figure 6, the total execution time is 27000 ps or 27 ns with a clock cycle time of 2 ns. Total number of clock cycles equals to 27 /2= 13.5. Therefore, CPI =13.5/7 ≈ 1.93. It runs 185.2% (77/27≈2.852) and 72.2% (46.5/27≈1.722) faster than the non-pipelined MIPS and the pipelined MIPS implementation with data dependencies respectively in term of execution time.

In summary, the pipelined MIPS is executed much faster than the non-pipelined MIPS in term of total execution time. Data dependency between consecutive instructions significantly affects performance of the pipelined MIPS implementation. Data dependency can be avoided or alleviated by employing data forwarding, instruction reordering and other techniques.

5. Conclusion

In this paper, a MIPS processor design and its front end verification have been successfully accomplished with pipelining mechanism. The observed CPI is 1.93 and can be considered as an efficient design when compared to the pipelined MIPS with data dependencies where a CPI of 3.32 has been spotted. The design also runs 1.852 times and 72.2% faster than the non-pipelined MIPS and the pipelined MIPS with data dependencies respectively. An enhancement to the design like a cache structure has also been used in order to replace the data memory.

6. References

[1] Hennessy & Patterson, *Computer Architecture: A Quantitative Approach (5th edition)*. The Morgan Kaugmann Series in Computer Architecture and Design, 2012.

[2] I. Pantazi-Mytarelli. The history and use of pipelining computer architecture: MIPs pipelining implementation Systems. *IEEE Proc. of Applications and Technology Conference (LISAT)*, 2013, pp. 1-7.

[3] K. Lasith, & A. Thomas,. Efficient implementation of single precision floating point processor in fpga. *Proc. of Emerging Research Areas: Magnetics, Machines and Drives (AICERA/iCMMD).* 2014, pp. 1-5

[4] D. Williamson, S. Steps, & B. Thompson.1984. Designing a low-cost 3-mips computer. *Hewlett-Packard Journal.* 1984, 35(2): 12-17.

[5] D. Kumar, & K. Singh. Design of High Performance MIPS-32 Pipeline Processor. 2012.

[6] K. Singh, S. Parmar, & A. Professor. (2012). Vhdl Implementation of a Mips-32 Pipeline Processor. 2012.

[7] InternetResource:http://minnie.tuhs.org/CompArch/Lectures/week02.html

[8] K. Kedzierski, M. Moreto, F. Cazorla, & M. Valero. (2010). Adapting cache partitioning algorithms to pseudo-lru replacement policies. *IEEE Proc of Parallel & Distributed Processing (IPDPS), 2010, pp.* 1-12.