EFFICIENT ARCHITECTURE TO ENHANCE 32-BIT PIPELINED PROCESSOR

Praveen Pilla, MD Khwaja Muinuddin Chisti, Koteshwaraonaik R

Department of Electronics and Communication Engineering
praveen06889@gmail.com, chist09@gmail.com, kotimal@gmail.com

Abstract— This paper deals with reduced hardware for eliminating branch penalty cycles in pipelined processor. Generally control hazard causes a greater performance loss in pipelined processor such as MIPS. Here we present improved pipelined processor architecture as well as eliminating branch and jump penalty. In the proposed architecture the number of multiplexers and adders were reduced compared to previously implemented architecture. The proposed architecture is implemented in Xilinx 10.1 tool using Verilog.

Index Terms— Branch address, Branch penalty, Control hazard, Dual port memory, Verilog

I. INTRODUCTION

Now a days Integrated Circuits (ICs), is becoming increasingly important as designs become more and more complicated. It is important to achieve a high level of reliability with minimum cost and time. The recent trend shows the RISC processors is clearly outsmarting the earlier CISC processor architectures. The reasons have been the advantages, such as its simple, flexible and fixed instruction format and hardwired control logic, which paves for higher clock speed, by eliminating the need for microprogramming. The combined advantages of high speed, low power, area efficient and operation-specific design possibilities have made the RISC processor ubiquitous.

RISC processor supports Pipelining design to increase the performance. For efficient pipelining implementation the following tasks concerned: Use multi-cycle methodologies to reduce the amount of computation in a single cycle. The Shorter computations per cycle allows for faster clock cycles. Overlapping instructions allows all components of a processor to be operating on a different instruction. Throughput is increased by having instructions complete, more frequently. In general, five stages pipelining includes Instruction Fetch(IF), Instruction Decode(ID), Execution(EX), Memory(MEM), and Write Back(WB) [1,3]. In addition to these stages, each stage is separated by special registers called pipeline registers. The purpose of these registers is to separate the each stage of the instructions so that there is no conflicting data due to multiple instructions being executed simultaneously. These pipeline registers are named after the stage that they are placed in-between: IF/ID Register, ID/EX Register, EX/MEM Register, and MEM/WB Register.

Cycles	i	i+1	i+2	i+3	i+4	i+5	i+6	i+7	i+8
i	IF	ID	EX	MEM	WB				
i+1		IF	ID	EX	MEM	WB			
i+2			IF	ID	EX	MEM	WB		
i+3				IF	ID	EX	MEM	WB	
i+4					IF	ID	EX	MEM	WB

Fig. 1: Process of pipelining.

The pipeline execution process is shown is Fig. 1. For every clock cycle an instruction is fetched. During this pipeline process hazards are produce such as structural hazard, data hazard and control hazard.

Structural hazard occurs when the hardware cannot support the combination of instructions that are set to execute in the given clock cycle. Data hazard occurs when the data that is needed to execute the instruction is not yet available.

Control hazard occurs when the instruction that was fetched is NOT the one that is needed; that is, the flow of instruction addresses is not what the pipeline expected. Control hazard also called as branch hazard which complicates the flow of instructions into the pipeline that depends on the condition whether a branch to be taken or not taken. If a branch is not taken, the flow of control is unchanged and the next instruction to be executed is the instruction immediately following the current instruction in memory; if a branch is taken, the next instruction to be executed is the branch destination instruction. While executing a branch instruction, processor needs to calculate the branch's destination address. To calculate the branch address there is wastage of atleast one clock cycle.

To eliminate the branch penalty, pipelined processor architecture is modified by calculating branch address in advance along with program counter value [4]. But to do so, more number of multiplexers and adders were used which increases the hardware components and power requirements.

In order to overcome the hardware requirements we have come out with a modified design. In section 2 we have presented our proposed modified architecture which reduces the hardware components. The simulation results and the performance shown in this paper are similar with [4], but with reduced hardware requirements.

II. PROPOSED ARCHITECTURE

This section presents the proposed novel architecture which eliminate branch and jump penalty. Branch address is calculated as:

Branch Address= (Offset<< 2) + PC + 4;

Format for branch instruction used in this paper is shown in Fig. 2.

Opcode	Rs	Rt	Offset
6 bit	5 bit	5 bit	16 bit

Fig. 2: Branch instruction format (32bits)

Where Rs: source register, Rt: Target register.

Here we present two port instruction memory as in [4], whose two output instructions correspond to program counter(PC) and branch destination address locations respectively. Depending on the branch condition one of the instructions is fetched. The selection of instruction is done by placing a multiplexer in between instruction memory and ID/EX register. The proposed architecture is shown in Fig. 3.

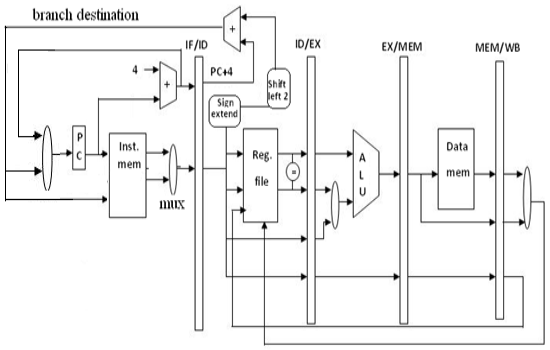

Fig. 3: Proposed pipelined Architecture.

The pseudo code for the proposed method is shown in Fig. 4. Nextpc is the register always updated concurrently with either program counter (PC_Addr) or branch destination address (Branch_Addr) depending upon branch condition (PCSrc). If branch condition is true then PCSrc is considered as '1'. Therefore Nextpc will have the address value of branch address. PC will be always updated for clock positive edge event. Thus by the time PC takes the new address value Nextpc will be updated with new branch address, if branch condition is true. The output of the dual port instruction memory consists of two instructions depending upon the PCSrc value. One of the instructions is selected by using the same PCSrc signal through the 2x1 Mux as shown in the proposed architecture as shown in Fig. 3. i.e the fetched instruction by the program counter (inst_pc) or by the branch destination address (inst_dist) will be considered to be fetched and execute.

```
assign nextpc = PCSrc ? Branch_Addr : PC_Addr;
always @ (posedge clock)
begin
if(PCWrite)
begin
pc = nextpc;          //update pc
end
end
//-------------------
InstructMem IM(pc,BranchAddr, Inst_pc, inst_dist);
assign IFinst = PCSrc ?  inst_dist :  Inst_pc;
```

Fig. 4: Pseudo code for proposed architecture

The simulation results of the proposed architecture and previous architecture [4] are shown in Fig. 5 and Fig. 6. The branch address calculation and comparison of the registers are done simultaneously in the first half of clock cycle. The Nextpc is updated with (branch address) + 4 when the PCSrc is true in the second half of clock cycle. The results shows similar performance [4] with reduced hardware cost.

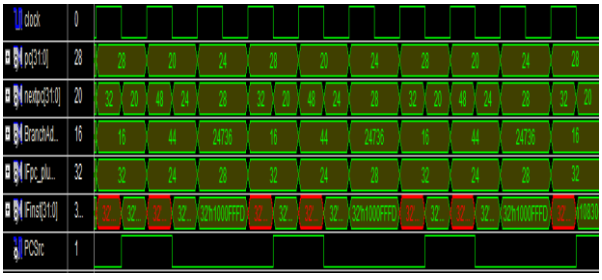

Fig. 5: Simulation results of proposed architecture.

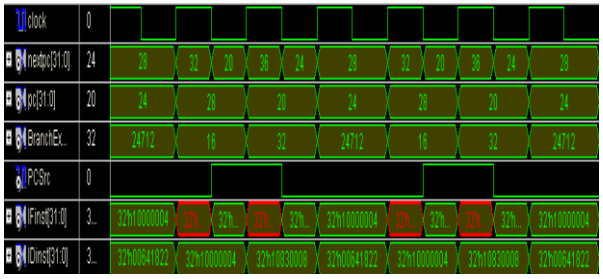

Fig. 6: Simulation results of proposed architecture.

III. SYNTHESIS REPORT

The proposed architecture and previous architecture are synthesized using Xilinx 10.1. The synthesis report of proposed architecture and previous architecture [4] are shown in Fig. 7 and Fig. 8 respectively. Here we considered instruction memory to have 512 memory locations. From the synthesis report it is concluded that number adders were reduced significantly.

```
Advanced HDL Synthesis Report

Macro Statistics
# Multipliers                          : 1
 16x16-bit multiplier                  : 1
# Adders/Subtractors                   : 4
 32-bit adder                          : 3
 32-bit subtractor                     : 1
# Counters                             : 1
 32-bit up counter                     : 1
# Registers                            : 2112
 Flip-Flops                            : 2112
# Latches                              : 1
 4-bit latch                           : 1
# Comparators                          : 7
 32-bit comparator equal               : 1
 5-bit comparator equal                : 6
# Multiplexers                         : 5
 32-bit 32-to-1 multiplexer            : 3
 32-bit 4-to-1 multiplexer             : 2
```

Fig. 7: Synthesis report for proposed Architecture

```
Advanced HDL Synthesis Report

Macro Statistics
# Multipliers                          : 1
 16x16-bit multiplier                  : 1
# Adders/Subtractors                   : 14
 32-bit adder                          : 9
 32-bit subtractor                     : 5
# Counters                             : 1
 32-bit up counter                     : 1
# Registers                            : 2112
 Flip-Flops                            : 2112
# Latches                              : 1
 4-bit latch                           : 1
# Comparators                          : 7
 32-bit comparator equal               : 1
 5-bit comparator equal                : 6
# Multiplexers                         : 6
 16-bit 4-to-1 multiplexer             : 1
 32-bit 32-to-1 multiplexer            : 3
 32-bit 4-to-1 multiplexer             : 2
```

Fig.8: Synthesis report for previous Architecture

The numbers of adders used in previous architecture were fourteen where as in proposed architecture the number of adders was three. The numbers of multiplexers used in previous architecture were six where as in proposed architecture the number of multiplexers was five.

IV. CONCLUSION

In this paper the proposed modified architecture reduces the hardware requirements such as adders and multiplexers without compromising the performance by eliminate the branch penalty. Simulation and synthesis results of proposed architectures were produced in Xilinx 10.1 tool using Verilog HDL.

REFERENCES

[1] Anjana R & Krunal Gandhi," VHDL Implementation of a MIPS RISC Processor," International Journal of Advanced Research in Computer Science and Software Engineering, Volume 2, Issue 8, August 2012.

[2] C. Perleberg and A. J. Smith, Branch Target Buffer Design and Optimization, IEEE Trans. Computers, volume 42, p. 396-412, 1993.

[3] David A. Patterson, and John L Hennessy. 2006,Computer Architecture A Quantitative Approach, 4th Edition;.

[4] Md. Raqibul Hasan, M. Sohel Rahman, Masud Hasan, Md. Mahmudul Hasan, M. Ameer Ali, "An Improved Pipelined Processor Architecture Eliminating Branch and Jump Penalty," Second International Conference on Computer Engineering and Applications, 2010.

[5] G. Kane, MIPS RISC Architecture, Prentice Hall, 1989

[6] S. McFarling, J. HenneseyReducing the cost of branches, Proceedings of the 13th annual international symposium on Computer architecture (ISCA '86) Pages: 396 – 403.

◈◈◈