Selective branch prediction schemes based on FPGA MIPS processor for educational purposes

To cite this article: H S Mahmood and S S Omran 2019 IOP Conf. Ser.: Mater. Sci. Eng. 518 042008

View the article online for updates and enhancements.
Selective branch prediction schemes based on FPGA MIPS processor for educational purposes

H S Mahmood1*, S S Omran1
1Electrical Engineering Technical College, Middle Technical University, Baghdad, Iraq
*Corresponding another: e-mail: hadeel_shakir@yahoo.com

Abstract. Processor performance is measured by amount of ILP (Instruction Level Parallelism) represented by its design. this parallelism is limited by the execution of conditional branch instructions which may break the flow of the program execution. To overcome this problem, several ways were suggested in order to predict both the direction of instructions execution and the address of the instruction to be executed next. In this paper, a design for a dynamic branch predictor was implemented in VHDL (VHSIC Hardware Description Language) then it was integrated with FPGA (Field Programmable Gate Arrays) MIPS (Microprocessor without Interlocked Pipelined Stages) processor and its ability to increase prediction accuracy and MIPS processor performance was approved. This predictor combines gshare and bimodal branch prediction techniques by dividing the PHT (Pattern History Table) into two branch streams corresponding to taken and not-taken states. This combined branch predictor was synthesized using the XILINX ISE (Integrated Software Environment) Design Suite 14.7 tool.

1. Introduction
In all processors that using pipelined stages, the performance is affected by the execution of some instructions that need to stall the pipeline for several clock cycles in order to get the correct results. These stalls in turn will increase the value of CPI (Cycle Per Instruction) and reduce the overall processor performance. The most common instructions in effecting the flow of the program execution are the branch instructions. If the instruction to be executed is a branch, more clock cycles are needed by the processor not only to know whether the branch will be taken or not, but also to calculate the value of either the target address in the case of a branch is taken or simply the next address in the case of a branch is not taken. Hence branch prediction (BP) is traditionally used to solve this branch problem. The BP could be static (always fixed), or dynamic, which means it is changing its prediction in run time. The Intel i486 used static BP that always not taken approach [1]. While the Pentium_M processor uses a dynamic BP [2].

In this paper, a study and design for three techniques of dynamic branch predictors were implemented using a previously designed pipelined MIPS processor for educational purposes. The dynamic BP needs more resources of hardware as comparison with the static BP which is very easy and simple to implement in hardware, the three BP techniques were used in this paper are the bimodal BP, the gshare BP and the combined BP.

Following sections are organized as follows. Section 2 discusses some previous works related to branch prediction, whereas later sections represent the VHDL implementation and configuration of the combined BP design. Firstly, the branch prediction principles are reviewed in section 3. Secondly, the design aspects are discussed in section 4. Thirdly, the FPGA configuration is shown in section 5. Finally, the gotten results and conclusion are given in sections 6 and 7 respectively.
2. Related work
In general, when using the BP, it will reduce the latency and improves the processor performance [3]. For example, reducing the miss prediction by half improves the processor performance by 13% [4]. Some researchers studied the static or fixed branch prediction which is very simple to implement in hardware [5]. While other researchers used dynamic branch predictor which will achieve higher prediction rates in the range of 80-90% [6, 7, 8].

The bimodal branch prediction is the early one used by several researchers like [4, 9, 10], they found that this method gives a prediction of 92% correct. Some other researchers used the gshare branch prediction technique [6, 11], and [12] used a branch address global history, while [13] proposes a combination of two branch predictors the gshare and bimodal where the two levels of prediction that used by these two methods give an improvement of 93.5% in prediction accuracy.

3. Branch prediction principles
In normal circumstances, a processor will execute the program instructions sequentially until a branch instruction is encountered. This branch instruction will be either taken or not taken. When taken, it instructs the processor to change the sequence of program execution and move to a new target address. When not taken, the execution will continue as no branch occurred.

As a result, a branch predictor is needed to guess both the direction of the branch (taken or not taken) and the target address without having to wait for the branch instruction to be executed. First, an always not taken static PB was used in MIPS processor to reduce the control hazard that happens when a branch is needed. An equal unit is placed in the decode stage where the branch decision is made. According to this decision, either the processor should flush the decode register and start fetching instructions from the branch target address when the branch is taken, or it should continue the sequential program execution when the branch is not taken [14].

Then a dynamic BP is used to predict the direction and the target address depending on an information collected during the program execution. This BP is arranged in two types, local and global prediction. In next sections, a VHDL implementation of a bimodal, gshare and combined BP on an FPGA is illustrated.

4. Design aspects of branch predictor
The implemented BP lies on the fetch stage of the MIPS processor and has the following design aspects:

4.1. Target Address Calculation Unit (TAC)
This unit is placed in the fetch stage and used to calculate the instruction’s target address. During the fetch stage, the processor reads the instruction from the instruction memory and passes it along with the PC value to the TAC unit which in turn calculates the target addresses for each of next PC (pcplus4), direct branch (pcbranch), jump (pcjump) and indirect branch (pcjr) as shown in figure 1. Then depending on whether the predict is taken or not the required target address is selected and passed to PC register.

Putting the TAC unit in the fetch stage will improve accuracy and eliminate the need for BTB (Branch Target Buffer) since the address provided by this unit is always correct.
4.2. Bimodal BP unit

This technique predicts the direction of the current branch instruction based on its history. As shown in figure 2, this Bimodal BP consists of a Pattern History Table (PHT) that holds 1024 entries. As in figure 3, each entry is a 2-bit saturating counter which is set to a default value of (11) and is updated depending on whether the current branch is taken or not in real [15].

Figure 1. Target Address Calculation Unit (TAC).

Figure 2. Bimodal BP unit.
The PHT is placed in the fetch stage and indexed by the lower bits of the branch address (pcF[11:2]), the upper bit of the indexed counter (predictF[1]) whatever it is ‘0’ or ‘1’ is used to predict the branch direction as not taken or taken respectively. Later in the decode stage, the entry corresponding to that branch address (pcD[11:2]) is updated depending on its real direction (pcs_D) which become available after resolving the branch instruction as in figure 2. Although this technique is shown to be effective in improving the accuracy of branch instruction execution, it may suffer from aliasing that happens when a predictor entry is used by multiple unrelated branches.

**Figure 3.** 2-bit saturating counter.

### 4.3. Gshare BP unit

In order to reduce the aliasing problem caused by the local prediction, a global BP technique is used. The gshare BP, shown in figure 4, predicts the direction of the current branch based on the history of the last few branches which is held by a shift register called Global History Register (GHR). In gshare BP, the local branch address pcF(11:2) is xored with the GHR(9:0) and used to index the PHT and the prediction of the current branch is read in the fetch stage.

As in bimodal BP, the branch instruction is resolved in the decode stage and according to its real direction (pcs_D) the entry indexed by (gshrD) is updated.
4.4. Combined BP unit

A more accurate BP based on combining different BPs and take their advantages at different conditions. Figure 5 shows that the combined BP is formed by combining the bimodal BP and gshare BP with an additional PHT containing 1024 entries each holds a 2-bit saturating counter. This additional PHT is indexed by the lower part of the branch address pcF(11:2) in the fetch stage and (Tourslct), the MSB (Most Significant Bit) of the 2-bit counter in the indexed entry, is used to select which predictor is more suitable for the branch, where P1 is selected if it is ‘1’ otherwise P2 is selected.

After resolving the branch later in the decode stage the counter will be updated as in figure 5. The counter is either incremented as P1 is right and P2 is wrong, decremented as P2 is right and P1 is wrong or not changed as both P1 and P2 have the same prediction whatever it was right or wrong.

**Figure 4.** Gshare BP unit.
5. Branch predictor configurations on an FPGA
The hardware models of the dynamic BP schemes were first implemented using a pure VHDL code, then from this code, the RTL schematic modules have been introduced using RTL Viewer of XILINX ISE Design Suite 14.7. Figure 6 shows the schematic module of the combined BP.
Figure 6. Schematic module of the combined BP.

6. Results
After completing the hardware modelling of the three dynamic BP schemes explained earlier, four test programs with branches as 20% of all instructions were executed on a 32-bit pipelined MIPS processor using either always not taken static BP or one of these three dynamic BPs, then their results were compared in term of program execution time to show the effectiveness of each BP scheme used as in table 1.

Table 1. Comparison between different BP schemes in term of program execution time.

| BP scheme          | Test program1\(^a\) | Test program2\(^b\) | Test program3\(^c\) | Test program4\(^d\) |
|--------------------|---------------------|---------------------|---------------------|---------------------|
| Always not taken   | 250 ns              | 450 ns              | 314 ns              | 470 ns              |
| static BP          |                     |                     |                     |                     |
| Bimodal BP         | 218 ns              | 410 ns              | 270 ns              | 410 ns              |
| Gshare BP          | 238 ns              | 410 ns              | 290 ns              | 438 ns              |
| Combined BP        | 206 ns              | 410 ns              | 258 ns              | 408 ns              |

\(^a\) Test program1 repeating a loop for 20 times.

\(^b\) Test program2 finds the Factorial of 100 by calling a function that contains loop and some unconditional branches.

\(^c\) Test program3 performs some conditional branches.

\(^d\) Test program4 combining loops with some conditional branches.

In general, number of branches in each program = 20% of its instructions.
As it is clear in table 1, test program 2 takes 410ns when executed in all dynamic BP schemes since it contains only function call instructions and unconditional branches which are always taken instructions and cause no direction miss predict. For test programs 1, 3, and 4, in spite of the aliasing problem from which bimodal BP suffers, its performance is still good for a 1024 PHT. Although the aliasing problem was reduced by using the gshare BP, its performance is not better than the bimodal BP for same test programs. Combined BP shows that it gives the best prediction accuracy since it takes advantage of local behaviour provided by the bimodal scheme and use global behaviour of the gshare BP to reduce the aliasing problem. Figure 7 represents the simulation waveforms gotten when test program 4 was executed by each BP scheme using the XILINX ISE Design Suite 14.7 tool.

Figure 7. Simulation results of test program 4.

7. Conclusion

Several schemes of branch prediction were implemented in this research using VHDL and their performance was presented for educational purposes. These hardware models used in MIPS processor to execute several test programs that have branches up to 20% of its instructions and the desired results were obtained. Starting with always not taken static BP which has the simplest design with a less performance accuracy and ending with three dynamic BP schemes, the results illustrates that the local bimodal BP has a good prediction accuracy with an increasing sufferance from aliasing while the global gshare BP reduces aliasing probability by exploiting the history of the last few branches to predict the next branch direction. The best result can be achieved by combining local BP with global BP. In spite of its complicated design, combined BP shows the best prediction accuracy.
8. References

[1] Corporation I 1997 Embedded Intel486 Processor Hardware Reference Manual.

[2] Simcha Gochman, M P G, Intel Corporation, et al. 2003 The Intel® Pentium® M Processor: Microarchitecture and Performance. May 21, 07(02).

[3] Seznec A, Felix S, Krishnan V, and Sazeides Y 2002 Design tradeoffs for the Alpha EV8 conditional branch predictor ISCA pp 295-306.

[4] Jiménez D 2011 An optimized scaled neural branch predictor ICCD pp 113-118.

[5] Smith J 1981 A study of branch prediction strategies ISCA pp 135-148.

[6] Loh G 2005 A simple divide-and-conquer approach for neural-class branch prediction PACT pp 243-254.

[7] Gao H, Ma Y, Dimitrov M, and Zhou H 2008 Address-branch correlation: A novel locality for long-latency hard-to-predict branches HPCA pp 74-85.

[8] Amant R, Jiménez D, and Burger D 2008 Low-power, high-performance analog neural branch prediction MICRO pp 447-458.

[9] Milenkovic M, Milenkovic A, and Kulick J 2002 Demystifying intel branch predictors Workshop on Duplicating, Deconstructing and Debunking 2002.

[10] Lee C, Chen I, and Mudge T 1997 The bi-mode branch predictor MICRO pp 4-13.

[11] McFarling S 1993 Combining branch predictors Digital Western Research Laboratory Tech. Rep.1993.

[12] Seznec A and Michaud P 1999 De-aliased hybrid branch predictors Ph.D. dissertation INRIA 1999.

[13] Eden A and Mudge T 1998 The YAGS branch prediction scheme MICRO1998 pp 69-77

[14] Omran S and Mahmood H 2014 VHDL Prototype of a 5-Stages Pipelined RISC Processor for Educational Purpose MESM’2014.

[15] Hennessy L and Patterson D 2011 Computer Architecture: A Quantitative Approach, 5th (Morgan Kaufmann Publishers Inc. San Francisco, CA, USA).