Journal of Nano- and Electronic Physics
Vol. **12** No 5, 05021(4pp) (2020)

Журнал нано- та електронної фізики
Том **12** № 5, 05021(4cc) (2020)

Design and Development of an Efficient Branch Predictor for an In-order RISC-V Processor

C. Arul Rathi[1,*], G. Rajakumar[1], T. Ananth Kumar[2,†], T.S. Arun Samuel[3]

[1] *Francis Xavier Engineering College, Tirunelveli 627003, India*
[2] *IFET College of Engineering, Tamilnadu 605108, India*
[3] *National Engineering College, Kovilpatti 628503, India*

(Received 23 June 2020; revised manuscript received 15 October 2020; published online 25 October 2020)

Conditional branches are a serious issue in the pipelined processor. The branch direction and branch target address are determined and calculated by the processor after several cycles of the instruction decode, which results in the pipeline stall. Pipeline stall leads to control hazards in the processor and results in performance degradation. To increase the rate of the instruction flow in modern processors, branch prediction is used. Branch prediction provides an ideal speedup in performance of the processor. The processor predicts the direction in the branch prediction and determines instructions in accordance with the predicted path. The processor tests any prediction for the branch when the branch condition is calculated. If the prediction is incorrect, the processor will automatically abort all instructions taken along the wrong path and return the state to the address of the determined branch. An inaccurate branch predictor results in increased program run-time and leads to higher power consumption. Once the position of a branch is known, the actual target address of the next instruction must also be determined along the expected path. If the branch is expected not to be taken, the destination address is simply the address of the current branch plus the size of the command word. Unless the branch is to be taken, then the target depends on the branch type. The branch target buffer (BTB) can reduce branch efficiency by predicting the branch path and storing information used by branch. There are no stalls if the branch entry is found in BTB, and the calculation is accurate, or the penalty shall be two cycles or more. This paper focuses on the design and development of branch predictor with BTB for the fetch unit, which further integrates to an in-order pipelined RISC-V processor. The performance of the RISC-V core in terms of clock cycle latency, instruction per cycle (IPC), was measured and analyzed.

Keywords: Branch target buffer, Pipeline, Hazard, Branch predictor, Fetch, Conditional and unconditional instruction.

DOI: 10.21272/jnep.12(5).05021 PACS numbers: 93.85.Tf, 91.30.pd

1. INTRODUCTION

RISC-V is a modern instruction set architecture with standard open architecture which is designed to be scalable for a wide range of applications [3]. To design high-performance systems [8], the speed of operations called throughput and the number of calculations per unit time become essential [7]. We often go for a technique called pipelining. Pipelining [1] is a standard feature used in RISC-V processors. Pipelining involves not only executing instructions over multiple cycles but also executing multiple instructions per cycle, i.e. we are going to overlap instructions. Every phase of the pipeline is known as a stage. The five-stage pipeline processor is implemented [11] (fetch, decode, execute, memory, write back).

The primary motivation for having a pipeline is that our instruction is arriving continuously once every cycle. So, we should be able to feed one new instruction every cycle, which is fetched and then executed moving from one stage to the other IF, ID, EX, MEM, WP. So, a new instruction should be available for every cycle of the pipeline; otherwise, it leads to hazards in the pipeline. Hazards are basically classified into three types. The first kind of hazard can be a structural hazard. The structural hazard arises when two instructions are already in the pipeline in two different stages, but they are trying to use the same hardware resource. The second kind of hazard is called a data hazard. The data hazard can arise due to instruction dependency. The third kind of hazard is called a control hazard. This arises because of branch instructions.

For getting efficient output of the processor, it must fetch and decode instructions at a higher bandwidth. The branch predictor helps to improve the performance of pipelining or superscalar processors by predicting the branch at an early stage. The branch predictor prefects a minimal data structure and attempts to predict the branch output. The processor will assume the execution of an instruction based on the predicted result from the branch predictor.

A processor can have better overall performance, especially when the prediction rates are high [13, 14]. Without branch prediction, a processor must stop when branch instructions are not resolved. Further work on the question of forecasting conditional branches and instructions is written. This paper concerns the reduction of control hazards by developing an efficient branch predictor for an in-order RISC-V processor. Our proposed branch predictor includes branch target buffer (BTB) [4, 6], so the branch predictor can predict the direction of the branch as well as the branch target address. Further, this architecture design can be enhanced with a hybrid branch predictor and implemented with the RISC-V processor.

*ananth.eec@gmail.com

1.1 Existing System

Branch instructions can be problematic in a pipeline. This is due to a branch instruction, which informs the processor about the next instruction that focuses on another instruction. The branch instruction needs to execute another instruction, and according to the result, it decides to jump or not. The easiest method is always to assume the branch was not taken.

In this way, the correct instructions are exempted. If the assumption is correct, no action must be taken. If the assumption is incorrect, the pipeline will be flushed.

Branch prediction is another approach for the conditional divisions. In order to minimize pressures, it is expected that the branch will be taken or not taken in the early phase of the pipelining process. The forecast of the branch is divided into a static prediction and a dynamic prediction of the branch.

1.2 Static Branch Prediction

The static branch prediction is simple; it does not use any feedback from the run-time output. The static branch predictors are rule-based static branch predictors or profile-based static branch predictors.

Single-direction prediction. The single-direction prediction is the most straightforward branch prediction technique. In this case, the prediction is either taken or not always taken. The retroactive approach never implemented is the variation of the single-direction prediction.

Program-based prediction [12]. The compiler uses branch hints from the instruction set architecture to predict whether or not the branch is taken.

Profile-based prediction. The profile-based static predictors collect statistics from the instrumented version of a program and send the information to the compiler. This detail is used by the compiler as a branch hint for the final program.

1.3 Dynamic Branch Prediction

The dynamic branch predictors use the program execution information to predict the branch instruction. The dynamic branch predictors have high prediction rates than other techniques.

Smith predictor. Smith predictor [9] records all branches whether or not the branch is taken in the previous occurrence. The saturation counter reports when the branch happens and then raises the counter and reduces the counter if the branch is not taken.

Two-level predictor. The two-tier predictor [10] separates the branch history into the branch history record and pattern history table. The pattern history table lists the frequency of each occurrence of the branch. The content of the branch history record is used to index the pattern history table.

Bi-mode predictor. Multiple pattern history tables (PHTs) are used for the reduction of aliasing. In the bi-mode predictor [5], two PHTs are used, one to store the most taken branches, and the other is to store the most not taken branches. A choice predictor is used to choose between these two predictors.

YAGS predictor. The YAGS predictor [2] is identical to the bi-mode predictor, but two PHTs document only those instances that conflict with the direction bias.

More than one prediction method is used for a hybrid predictor, also called a combined predictor. To predict the branch, two or more branch predictors discussed above are combined.

2. PROPOSED SYSTEM

In the proposed system, the efficient branch predictor is designed for a RISC-V processor. The system gets the input from the fetch unit of the pipelined processor, and the generated output is given back to the same fetch unit. Another input comes from the execution stage, which is used to compare the predicted value and the actual executed value. The block diagram of the proposed efficient branch predictor is shown in Fig. 1.

The branch predictor consists of PHT and BTB, which are commonly used data structures. The PHT and BTB have been indexed according to the branch address. The PHT forecasts whether the branch is taken or not. The following instruction address is taken from the BTB when the branch is taken. The next command address if the branch is not taken is the current branch address plus instruction size.

Fig. 2 shows the architecture of an efficient branch predictor for the RISC-V processor. The PHT uses the 2-bit saturation counter state transition, which increases when the prediction is correct and decreases if the prediction is wrong. The PHT uses a valid bit to ensure the finishing of the training period. The valid bit will be initially set to zero. When the branch is encountered for the first time, it changes the valid bit to zero. The target branch address of the current branch instruction is stored in the BTB. The BTB is updated by the execution unit of the pipelined processor.

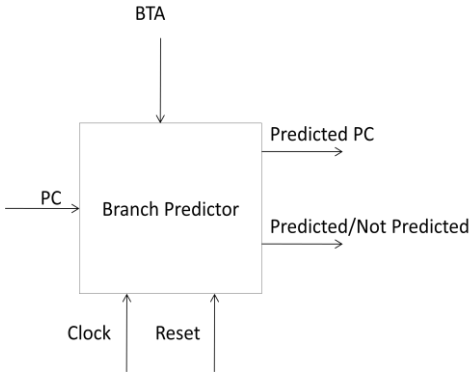

Fig. 1 – Block diagram of the branch predictor

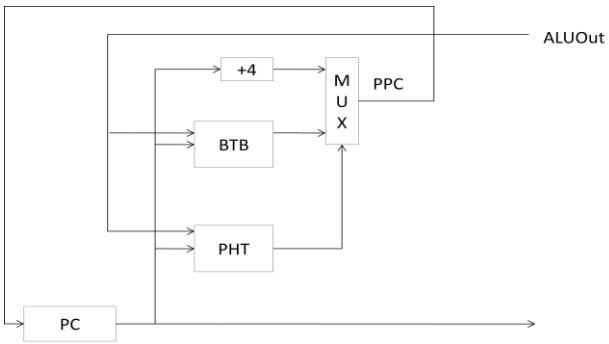

Fig. 2 – Architecture of the branch predictor

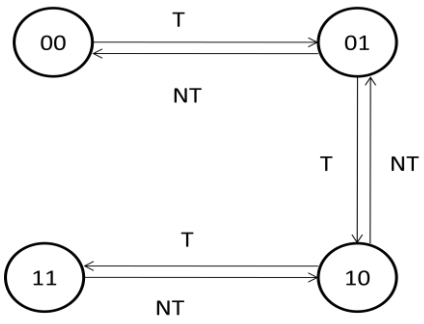

Fig. 3 – 2-bit saturation counter

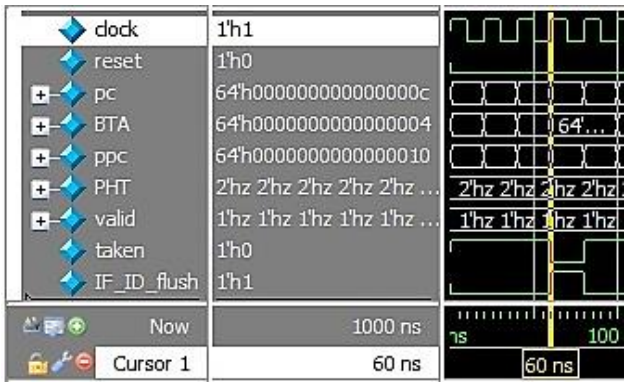

Fig. 4 – The branch instruction is identified in the branch predictor

The processor output is entered in the branch predictor. Then the branch predictor predicts whether or not the branch is taken. Finally, the predicted branch address is checked at the processor execution unit.

If it is correctly predicted, the execution unit gives a signal to the branch predictor that the prediction is correct. If it is wrongly predicted, then the execution gives a signal to the branch predictor that the prediction is incorrect. It also flushes out the incorrect values and goes with the executed correct value.

3. RESULTS AND DISCUSSION

Hardware describes the proposed model as HDL code and it is simulated in ModelSim. This hardware programming can be implemented successfully with Xilinx FPGAs. ModelSim is an HDL modeling environment that is multi-language. ModelSim may be used with or without Intel Quartus Prime, ISE Xilinx or Vivado Xilinx. The simulation is done through a graphical user interface (GUI) or using scripts automatically.

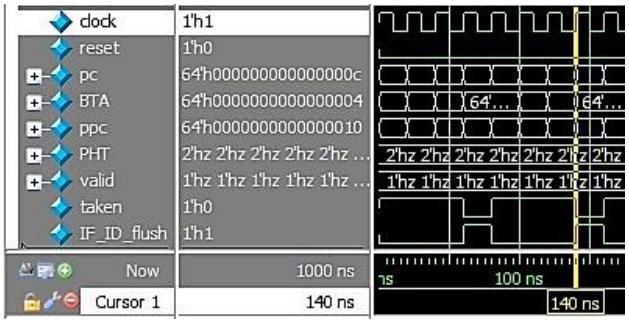

Fig. 5 – The branch predictor is trained in the run-time

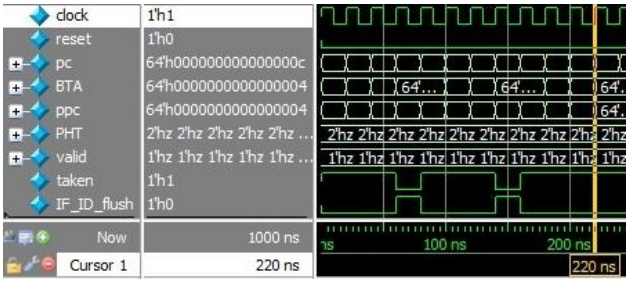

Fig. 6 – The branch predictor predicting the branch correctly

The proposed branch predictor is a dynamic branch predictor, so when the branch is encountered for the first time, it makes some note in the PHT. Fig. 4 shows the first appearance of the branch at a clock cycle of 60 ns.

The PHT uses the 2-bit saturation counter, so it will not predict the branch on the next encountering of the same branch unless it confirms the branch in the second appearance of the same branch. Fig. 5 shows the second appearance of the branch at 140 ns.

From the third time onwards, it will start predicting the branch correctly. Fig. 6 shows the branch predictor predicting correctly.

4. CONCLUSIONS

This paper explores the design and development of a branch predictor for a RISC-V processor. The branch predictor plays a major role in enhancing processor efficiency. The efficient branch predictor removes the control hazards in the pipelined processor. There is no need for static training period for the efficient branch predictor. The efficient branch predictor is designed successfully with a branch target buffer. It can be implemented with the in-order RISC-V processor to achieve higher performance.

REFERENCES

1. Chang, Po-Yung, Marius Evers, Yale N. Patt, *Int. J. Parallel Progr.* **25**, No 5, 339 (1997).
2. Eden, Avinoam N., Trevor Mudge, *In Proceedings. 31st Annual ACM/IEEE International Symposium on Microarchitecture* 69, (1998).
3. Evers, Marius, Po-Yung Chang, and Yale N. Patt, *Int. Symposium on Computer Architecture*, **24**, 3 (1996).
4. B. Fagin, K. Russell, *Int. Symposium on Microarchitecture*, 193 (1995).
5. Lee, Chih-Chieh, I-Cheng K. Chan, Trevor N. Mudge, *Int. Symposium on Microarchitecture*, 4 (1997).
6. Lee, K.F. Johnny, Alan Jay Smith, *IEEE Computer* **17** No 1, 6 (1984).
7. Scott McFarling, John L. Hennessy, *Int. Symposium on Computer Architecture*, 396 (1986).
8. Scott McFarling, *Compaq Computer Corporation Western Research Laboratory* (1993).
9. Jim E. Smith, *Int. Symposium on Computer Architecture,* 135 (1981).
10. Yeh, Tse-Yu, Yale N. Patt., *ACM SIGARCH Computer*

Architecture News **20**, No 2, 124 (1992).
11. Yeh, Tse-Yu, Yale N Patt, *Int. Symposium on Computer Architecture,* 124 (1992).
12. Yeh, Tse-Yu, Yale N. Patt, *Int. Symposium on Computer Architecture*, 257 (1993).
13. Kumar, T. Ananth, S.S. Janaki, *Int. J. Adv. Res. Eng. Technol.* **1**, No 3 (2012).
14. Kumar, T. Ananth, R.S. Rajesh, *International Conference on Communication and Network Technologies,* 254 (2014).