Branch Prediction

Anshul Choudhary
21110028

Anushk Bhana
21110031

Kaushal Kothiya
211100107

4 April 2023

Contents

1 Abstract

Branch prediction is an important technique used in modern processor design to improve performance. This project report aims to provide a basic understanding of branch prediction, including the need for a branch prediction unit in the processor systems, its advantages, the types of branch prediction, and the differences between branch prediction for architecture and organisation. To demonstrate the implementation of branch prediction, we designed a branch predictor for 3-stage sDLX processor. Overall, our project report provides a comprehensive understanding of branch prediction. Our design can be used as a reference for getting a good understanding of branch prediction for the optimisation of an sDLX processor.

2 Introduction

2.1 What is branch prediction?

Branch prediction is a convenient method to optimise the performance of a processor by minimising pipeline stalls caused by conditional branches. When a conditional branch instruction is encountered, the processor must wait until the condition is assessed before deciding which path of instructions to execute next. This can cause a pipeline stall, lowering the processor's overall performance. This issue can be tackled by predicting the outcome of the conditional branch instructions. When the actual branch instruction is resolved, the processor can continue executing instructions along the projected route. This can lessen the number of pipeline stalls and increase the processor's performance [1].

2.2 Why do we need branch prediction?

The processor has to wait for the outcome of the last instruction executed to decide which instruction is to be executed next correctly. This creates performance inefficiency where the processor can not meet the demanding data access rate. This issue arises from the need to use branch prediction to improve the processor's performance.

By predicting the outcome of a branch, the processor can start fetching and executing the instructions that are most likely to be needed without waiting for the outcome of the branch. If the prediction is correct, the processor has already fetched and executed the instructions that follow the branch, and hence the performance is improved. If the prediction is wrong, the processor has to discard the incorrect results and fetch and execute the correct instructions, which takes some additional time, but the overall performance is still improved. In summary, branch prediction helps to improve the performance of the processor by reducing the time it takes to execute the branches [2].

2.3 What if we don't use a branch predictor?

If we do not apply branch prediction, the processor will have to wait for the branch result before resuming execution, resulting in a pipeline stall, also known as a bubble. This stall would delay the execution of future instructions, resulting in reduced performance. However, in certain cases, such as simple systems with few conditional branches with known branch outcomes, branch prediction may not be required. The performance effect of not applying branch prediction may be negligible in such circumstances [1].

3 Branch prediction for architecture vs Branch prediction for organisation

Branch prediction, in terms of computer organisation, is concerned with maximising processor performance by lowering the number of pipeline stalls caused by conditional branches. In this situation, static branch prediction is frequently utilised since it is simpler and uses fewer hardware resources than dynamic branch prediction. Branch prediction in computer organisation is often concerned with reducing the influence of conditional branches on processor performance while employing simple and efficient strategies[3].

Branch prediction in computer architecture is mainly concerned with the design of hardware components and algorithms capable of properly predicting the outcome of conditional branches. In this situation, dynamic branch prediction is often employed since it achieves more accuracy than static branch prediction but necessitates more complicated hardware and algorithms. In general, branch prediction in computer architecture is concerned with anticipating the outcome of conditional branches with the greatest possible accuracy while minimising the impact of pipeline delays on processor performance[3].

Generally, the distinction between branch prediction in computer organisation and branch prediction in computer architecture is primarily due to the level of abstraction and concentration. Computer organisation is concerned with the lower-level implementation of processors. In contrast, computer architecture is concerned with the design of hardware and algorithms that facilitate the execution of software on processors. Yet, all disciplines are ultimately concerned with boosting processor speed, and branch prediction is an essential strategy for doing so [3].

4 Types of branch predictions:-

4.1 Static branch prediction

Static branch predictors are simple to implement and take less hardware. This is the main advantage of static branch predictors[4]. Some examples are shown below.

4.1.1 Single Direction Prediction

Single-direction prediction is the simplest branch prediction strategy. In this strategy, we assume always the branch taken/not taken. The prediction is made during the fetch stage of the pipeline. If the prediction is right, the processor can continue executing the instructions without waiting for the branch's true result. But if the prediction is wrong, then the processor must discard all the instructions that were fetched after the branch instruction and then get the right instructions[4].

4.1.2 Backwards Taken Forward Not Taken

In this type of static branch prediction, the branch is supposed to be taken if the current value of the PC exceeds the branch target. The branch is not supposed to be taken if the branch target exceeds the current value of the PC. This strategy can be utilitarian in cases where the loops run multiple times before leaving the loop, for example, for loop[4].

There are some other types of static branch predictors, like profiling-based branch predictors[4], but the key idea remains the same.

4.2 Dynamic branch prediction

Dynamic branch prediction techniques use the processor's run time information and can respond to changing branch patterns. This method is beneficial as the performance is increased without profiling.

4.2.1 Smith's Algorithm

In most of the branch predictors, the processor takes note of the actual outcomes of the instructions and then uses it to determine the outcome whenever the same branch is encountered[5].

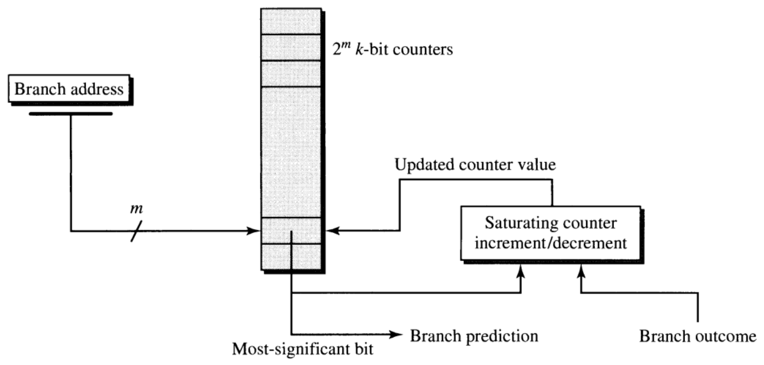
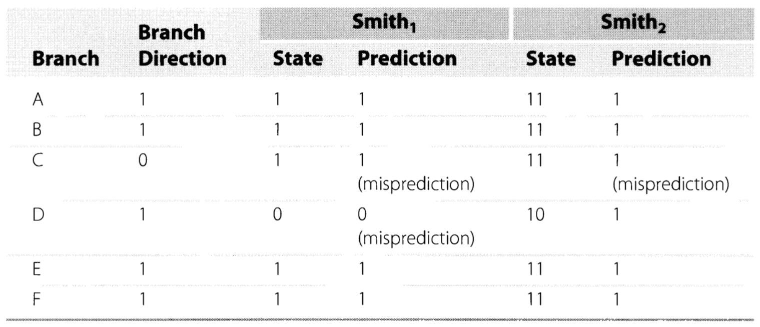


Figure 1 : Smith Predictor with a 2^m-entry Table of Saturating k-bit Counters[5].

In Smith's algorithm, we take m bits from the branch address, and according to that, we have a 2^m -entry table of k-bit counters. All the counters are initialised with 01. All the counters are directly mapped with the branch address.

We use the most significant bit for branch prediction. If the most significant bit is 1, then we predict the branch is taken; if the most significant bit is 0, then we predict that the branch is not taken.

If a branch is taken, we increase the counter by one saturated at $2^k - 1$. If the branch is not taken, we decrease the counter by one saturated at 0[5].



		Smith_1		Smith_2	
Branch	**Branch Direction**	**State**	**Prediction**	**State**	**Prediction**
A	1	1	1	11	1
B	1	1	1	11	1
C	0	1	1 (misprediction)	11	1 (misprediction)
D	1	0	0 (misprediction)	10	1
E	1	1	1	11	1
F	1	1	1	11	1

Figure 2 : Comparision of 2 Smith predictors with the single anomalous decision [5]

Here smith1 represents the smith's algorithm with value k=1.
Here smith2 represents the smith's algorithm with value k=2.

As we increase the value of k, hysteresis lag increases; for k=2 after the branch not taken instruction, it still predicts the branch taken. So, prediction doesn't change quickly as the value of k increases. As the value of K increases prediction accuracy increases[5].

4.2.2 Two Level Branch Prediction

In computer architecture, two-level branch prediction is a more sophisticated approach for predicting the outcome of conditional branch instructions. This strategy entails keeping a record of the behaviour of a certain branch instruction and utilising that record to create more accurate predictions[5].

There are two types of Two level branch predictors -

1. Global History two level predictor

The idea behind employing global branch history is that a branch's behaviour may be associated with that of an earlier branch. The most recent outcomes of the instructions are stored in Branch History Registers (BHR), which are n-bit shift registers in which the oldest outcome is discarded from the end. In this system, 0 and 1 represent whether the branch is not taken or taken, respectively. This is also known as the first level of global history two-level predictor[5].

The second level of this predictor comprises 2-bit saturating counters, which together form a table called Pattern History Table (PHT). This counter updates the same as in Smith's Algorithm, and the prediction is obtained from the MSB of the counter. The PHT is indexed by a concatenation of a hash of the branch address with the contents of the BHR[5].

The last m bits of PC are taken and are concatenated with the n bit BHR entry to form an $m + n$ bit index, this index is searched into the PHT, which has 2^{m+n} counters, and the corresponding counter to that index will determine the prediction in the similar fashion of the Smith's predictor[5].

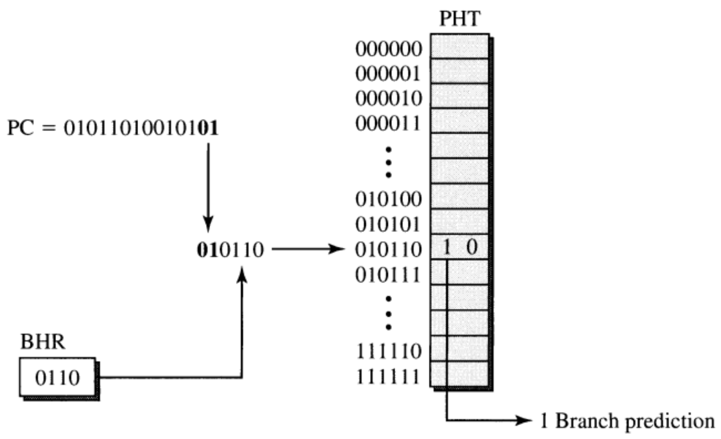


Figure 3 : A global history 2 level predictor with 4 bit BHR[5].

The only thing that restrains this method is the huge amount of hardware required to perform this prediction. If we were to take $m = 6$ and $n = 4$, then the number of counters in the PHT would be $2^{6+4} = 1024$ counters. This has some drawbacks as the $m + n$ bit index could be the same for multiple instructions as we are taking only m of the last bits from the PC and not the whole value[5].

2. Local History two level predictor

The difference between Local history and Global history is that global history contains the results of the last several branches and on the other hand, local history contains previous results only of the current branch[5].

The first level being the Branch History Table (BHT), one global BHR is replaced by one BHR per branch to construct a local-history branch predictor. A BHT is made up of several BHRs. The branch address is used to pick one of the BHT entries that contains the local history. To index into the PHT, the contents of the selected BHR are merged with the PC in the same manner as the global-history two-level predictor. The branch prediction is provided by the counter's most significant bit, and the counter's update is also the same as the Smith predictor[5].

Sizing a local-history two-level predictor involves more compromises than sizing a global-history predictor. There is a tradeoff between the number of bits dedicated to the BHT and the number of bits dedicated to the PHT, in addition to balancing the amount of history and address bits for the PHT index. If the number of bits of both BHT and PHT would be higher, we would receive much higher accuracy but the number of counters will be significantly high and there would always be size constraints[5].

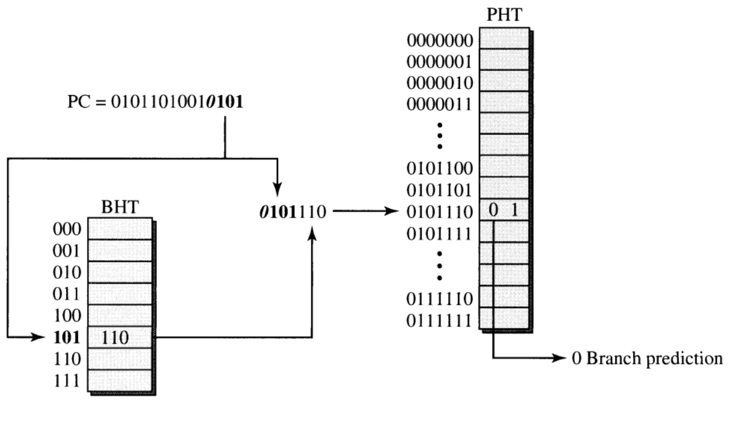


Figure 4 : A local history 2 level predictor with 3 bit history length and 8 entry BHT[5].

5 Comparison of newer and older branch prediction techniques

New branch prediction techniques	**Old branch prediction techniques**
These involve dynamic branch prediction methods such as Two level predictors, Hybrid predictors, Neural predictors, etc [5].	These involve static branch prediction methods like Single direction predictors, BTFNT predictors, profile based prediction etc [5].
Accuracy is much higher (83.4 percent to 97.5 percent when using 2 history bits) [5].	Accuracy is less than that of newer techniques (55 percent to 80 percent when branch was predicted always taken/not taken) [5].
More hardware components (such as counters and registers) are required [6].	No/very less hardware is required [6].
Suitable method for more intricate processors [6].	Apt choice for simple processors [6].

6 Desiging branch prediction unit for sDLX 3 stage processor

When 3 stage pipeline is used for the sDLX processor, We can decrease the number of stalls in the pipeline by using a branch predictor. We can implement a 2-level branch prediction algorithm for the sDLX processor. In our design, we have implemented a 2-level global branch predictor.

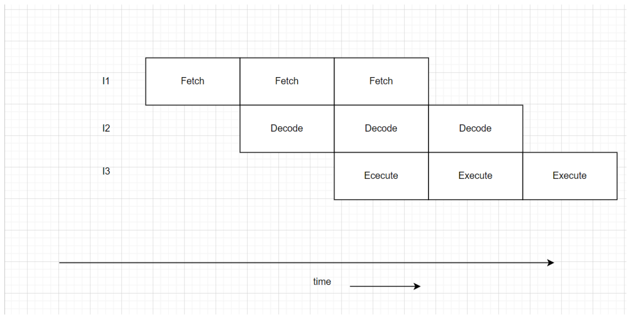


Figure 5 : 3 stage pipeline.

6.1 Working of the Design:-

1. At the Instruction Fetch level, we don't know if the instruction will be a branch instruction or not. So, we predict that it will not be a branch instruction.
2. At the Instruction Decode stage, if the instruction is not a branch instruction, then the prediction is correct. There are no stalls in the pipeline. If the prediction is wrong, i.e. the instruction is a branch instruction, then we will stop executing the instruction which comes after the branch instruction. There will be a stall in the pipeline if branch instruction comes.
3. When the Instruction decode is done, and if it is a branch instruction, then we predict whether the branch should be taken or not using the global branch prediction algorithm.
4. There is a enable signal for PC which is a "NOR" of 2 signals. One signal is for misprediction, and one signal will be for if the instruction is branch instruction or not. If the instruction is a branch instruction, then at the decode stage branch signal to the PC will be one, so PC won't be incremented.
5. We will have another register which stores the value of the PC and in case of misprediction we can have the correct value of the PC stored in the register.
6. If there is misprediction, then we will set the misprediction signal as 1. Then there will be another stall in the pipeline.
7. So, if it is a correct prediction, then there will only be one stall, and if it is a misprediction, then we have one more stall in the pipeline.
8. If it is a misprediction we have to correct the value in PHT.
9. After every branch instruction we have to change the value of BHR.
10. In case of Branch prediction or misprediction we will make Write enable 0 so that one instruction ahead of it will not execute. It will act as stalling in the pipeline. We can also say that we are flushing the next instruction.

6.2 Instruction for Python code

open the python file$->$ click on debugging$->$ click run python file
enter a number: loop will run till num<1000
Enter the value of RS1
Enter the value of the PC
Enter the value of the instruction
code will give the following outputs:
1. Branch taken/not taken
2. if there is a misprediction or not

3. BHR
4. PHT

7 Acknowledgement

We would like to express our sincere gratitude to everyone who helped us in completing this project report on branch prediction in ES215 Computer Architecture and Organization. Firstly, we would like to thank our professor, Sameer G Kulkarni, Assistant Professor of Computer Science and Engineering, for guiding us throughout the course and providing valuable feedback and insights. Your support and help have helped us explore and find an interest in a new topic. We also thank Professor Rajat Moona, Professor of Computer Science and Engineering, for allowing us to study branch prediction through a project.

We would also like to extend our thanks to our fellow classmates for their support and constructive feedback, which helped us improve the quality of our work. Finally, we would like to express our gratitude to the research papers, articles, and textbooks we referred to for our project. We are indebted to the authors of these sources for their hard work and dedication to advancing the field of Computer Architecture and Organization.

Thank you all for your support and guidance throughout this project.

8 Work distribution

Tasks Done	Time Taken	Done by
Report Making	2 days	Anshul, Anushk, Kaushal
Making of the Presentation	4 hours	Anushk
sDLX design	1 day	Kaushal
Python Code	8 hours	Kaushal
Research involved in theoretical understanding of the core concepts	1 week	Anshul, Anushk, Kaushal

9 References

[1]D. A. Patterson, P. Alexander, and J. L. Hennessy, Computer organization and design : the hardware/software interface. Waltham, Ma: Morgan Kaufmann, 2012.

[2]“What is branch prediction?,” Educative: Interactive Courses for Software Developers. https://www.educative.io/answers/what-is-branch-prediction (accessed Apr. 03, 2023).

[3]“Differences between Computer Architecture and Computer Organization - GeeksforGeeks,” GeeksforGeeks, May 13, 2019. https://www.geeksforgeeks.org/differences-between-computer-architecture-and-computer-organization/

[4]A review of branch prediction schemes and a study of branch predictors ... (no date). Available at: https://www.researchgate.net/profile/Venkata-Mekala/publication/266891966 A Review of Branch Prediction Schemes and a Study of Branch Predictors in Modern Microprocessors/links/545ac9ed0cf2c46f6643898c/A-Review-of-Branch-Prediction-Schemes-and-a-Study-of-Branch-Predictors-in-Modern-Microprocessors.pdf (Accessed: April 4, 2023).

[5]John Paul Shen and M. H. Lipasti, Modern Processor Design. Waveland Press, 2013.

[6] Branch prediction techniques used in pipeline processors : A Review (no date). Available at: https://acadpubl.eu/hub/2018-119-15/2/302.pdf (Accessed: April 4, 2023).

10 Additional Resources

We use these resources to learn about branch prediction. We would like to acknowlege the authors, publications for making our understanding better about the subject.

1. “Spring 2015 – Computer Architecture Lectures – Carnegie Mellon - YouTube,” https://www.youtube.com/playlist?list=PL5PHm2jkkXmi5CxxI7b3JCL1TWybTDtKq (accessed Apr. 04, 2023).

2. “Video 56: Bimodal Branch Predictors, CS/ECE 3810 Computer Organization” https://www.youtube.com/watch?v=Zeuc2PiXH8k (accessed Apr. 04, 2023).

3. D. A. Jimenez, “Piecewise linear branch prediction,” IEEE Xplore, Jun. 01, 2005. https://ieeexplore.ieee.org/document/1431572 (accessed Apr. 04, 2023).

4. Stallings, W. (2016) Computer Organization and Architecture: Designing for performance. Boston: Pearson-Prentice Hall.

5. Hennessy, J.L., Patterson, D.A. and Arpaci-Dusseau, A.C. (2007) Computer Architecture: A quantitative approach. Boston: Morgan Kaufmann.

6. J. Shen and M. Lipasti, "Fundamentals of Superscalar Processors." Available : http://acs.pub.ro/ cpop/SMPA/Modern Processor Design Fundamentals of Superscalar Processors(PDFDrive).pdf