# Branch Predictor Project Report

Pisit Wajanasara
PID: A59009987
University of California, San Diego

*Abstract*—**Branch instructions are part of modern computer programs in today world and are known to decrease the pipeline speed. To solve this problem, branch speculation was studied and various branch predictors were created to reduce the branch misprediction rate. In the early era, there are some predictors such as GShare, which can handle the global branch correlation, or the tournament predictor, which is composed of multi predictors and can handle both local and global branch correlation. In this work, we proposed a branch predictor based on the tournament predictor with some modification. Our result showed that it can beat both GShare and the tournament predictor for all testing traces in the term of misprediction rate.**

## I. Introduction

Real-world Computer programs consist of considerable amount of branch instructions. Without any branch speculation, the pipeline can be slowed down since it need to flush the pipeline in the case that fetched instructions were wrong. To boost the pipeline performance, modern computer archiectures increasingly rely on branch speculation in order to predict whether the branch will be taken or not. With more prediction accuracy, the pipeline can gain more performance.

The branch predictor is designed as a component which receive branch instruction address as its inputs. After processing, the branch predictor will output a bit to determine whether the given branch will be taken or not. One of the most advantage of the branch predictor is to reduce number of pipeline flush which improve the processor performance. However, designing the branch predictor come with limitations. First, the implemented component need to store inside the chip so it doesn't have a high latency.

## II. Background

## III. Design Ideas

Our design is based mostly on tournament branch predictor used in Alpha 21264's branch predictor.

In the tournament branch predictor, it used both local and global branch prediction with a choice prediction to choose the right predictor for different branches. This help the predictor to handle both local and global correlation between branches. In this work, we also utilize this idea and we decided to improve both local and global branch prediction method in our custom predictor.

First, we decided to look at the global predictor. If we compare between Alpha 21264's global predictor and GShare, we will notice that the index hashing schemes are different. For GShare, it used $index = (pc \oplus GHR) \mod n$ to index the global counter. While the latter predictor used only $pc \mod n$ as the index. We believe that GShare's index is better at incorporate both pc and GHR information to select the counter. In addition, we have more storage for storing the data for branch predictor. Thus, we decided to increase the size of global predictor too.

Secondly, for the local branch predictor, we observed that the implementaion in the Alpha 21264's predictor performed very well with 2-bit saturating counter. Thus, we decided to keep the counter as is. However, to efficiently use the increase storage size, we increased the size of the local history table instead.

The predictor chooser in the Alpha 21264's worked fine. However, we thought that the hashing index should be changed. Instead of indexing the chooser by the global history register, we changed the index to be the program counter instead. Thus, the chooser will select the predictor based on the current instruction address instead of the global history of the branch outcome.

## IV. Implementation details

In this section, we explained how our predictor was implemented. The proposed predictor is based on Alpha 21264's branch predictor which include the local predictor, the global predictor and the chooser as its components.

### A. Overall Design

The overview design of our predictor is shown in Figure 1. We have two different predictors inside the our predictor targeting global correlation and local correlation predictions. One thing to notice is that global predictor is using the xor result between the global history register and the program counter as the index. However, for the the rest of the components, we used only program counter for indexing.

During the prediction process, both predictors will give a prediction output, but only one prediction will be chosen by the predictor chooser. We incorporate a mux here as a selector. The output from the mux is the prediction result from our predictor.

After each prediction was done and the real branch outcome was revealed, the predictor need to be updated. Both local and global predictor are both updated at the same time with new information which we will describe the update process later in this report. The predictor chooser is also updated in this state. After all components are updated, we update the global history register to include the new outcome to our history. This is done by left shifting the global history register, then the new outcome is added to its last bit.

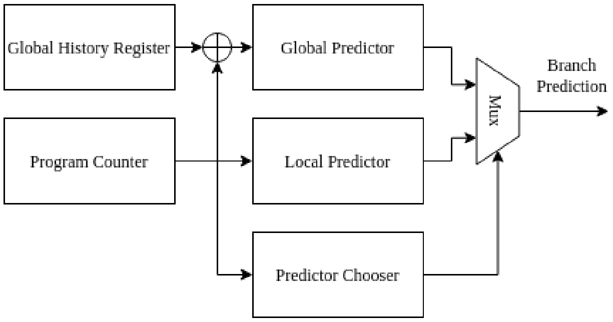
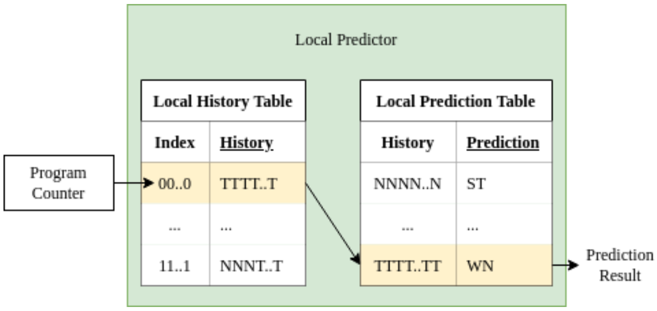


Fig. 1. The overall design of the proposed branch predictor. We have 2 predictors including the local and the global predictor for capturing different local/global branch patterns. The global predictor is indexed by the xored result between the global history register and the program counter. The local predictor is indexed by the program counter only. The predictor chooser select the result from both predictor by looking at the current address in the program counter.

### B. Local branch predictor

The presented local branch predictor in our work is based from the implementation in the tournament predictor. The n least significant bit from program counter is used to index the local history table. After that, we get the local history from that table. This history is a bit vector of Taken (T/1) and Not-Taken (N/0), which is used to index the local prediction table. The local prediction table will give the prediction result based on the state in the table. In this work, we used 2-bit predictors as our prediction table. This includes 4 possible states including strongly not-taken (SN/00), weakly not-taken (WN,01), weakly taken (WT/10) and strongly taken (ST/11). The first two states will give not-taken as the prediction result, while the last two will give taken as the prediction result. The internal process of this predictor is also shown in Figure 2.



| Local History Table | |
|---|---|
| Index | History |
| 00..0 | TTTT..T |
| ... | ... |
| 11..1 | NNNT..T |

| Local Prediction Table | |
|---|---|
| History | Prediction |
| NNNN..N | ST |
| ... | ... |
| TTTT..TT | WN |

Fig. 2. The local predictor has two components. First component is the local history table with hold the mapping table from address index to local branch history. Then, the obtained branch history is used to get the prediction table state which determines the prediction result.

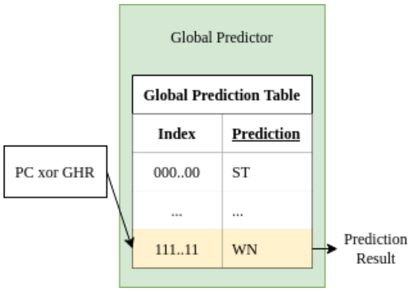
During the update phase, the history related to the instruction address is updated with the new outcome. The prediction table state also get updated regarding the new branch outcome. We update the state toward strongly not-taken and toward strongly taken when the outcome is not-taken and taken respectively. The update algorithm is shown in Algorithm 1.

**Algorithm 1** An algorithm with caption

```
Require: $n \geq 0$
Ensure: $y = x^n$
  $y \leftarrow 1$
  $X \leftarrow x$
  $N \leftarrow n$
  while $N \neq 0$ do
      if $N$ is even then
          $X \leftarrow X \times X$
          $N \leftarrow \frac{N}{2}$                ▷ This is a comment
      else if $N$ is odd then
          $y \leftarrow y \times X$
          $N \leftarrow N - 1$
      end if
  end while
```

### C. Global branch predictor

For the global branch predictor, we modified Alpha 21264's global predictor to use the GShare index instead of just the global history register. The GShare index is calculated by xoring the global history table with the address in the program counter. This help incorporate both information to select the right prediction state for the global prediction, i.e., the same address with a different history will point to a different row in the prediction table. Same as the local branch predictor, the global prediction table stores 2-bit states which related to strongly not-taken, weakly not-taken, weakly taken and strongly taken as described in the previous section. The internal implementation of this component is shown in Figure 3.



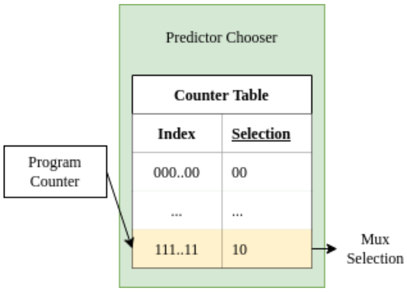
| Global Prediction Table | |
|---|---|
| Index | Prediction |
| 000..00 | ST |
| ... | ... |
| 111..11 | WN |

Fig. 3. The global prediction table is indexed using the xoring result between the program counter and the global history register. The prediction result is a 2-bit states which is the same as the prediction table used in our local predictor.

In order to update the global predictor,

### D. Predictor chooser

The predictor chooser is implemented in almost same way as the global branch predictor. The difference is that the chooser use the instruction address from the program counter as the index, and the 2-bit states determine the predictor instead. The state 00, 01, 10 and 11 are for strongly local predictor, weakly local predictor, weakly global predictor and strongly global predictor respectively. The implementation of the predictor chooser is illustrated in Figure 3.



| Counter Table | |
|---|---|
| **Index** | **Selection** |
| 000..00 | 00 |
| ... | ... |
| 111..11 | 10 |

Fig. 4. The predictor chooser is indexed by the address in the program counter. The state in the table determines which predictor to use. It acts as a selection bit for the output mux.

The chooser's state will be updated if the output from both predictors are different. We will decrease the counter state if the local predictor give the correct prediction and the state is not 00. In the other case, the counter is increased if the state is not 11.

### *E. Predictor Initialization*

For the local predictor, the local history table was initialize to all not-taken at every index. Moreover, we set the state for each row in the local prediction table to weakly not-taken state (01).

In the case of the global predictor, the global history register was also initialized to all not-taken. The state in each row is also set to weakly not-taken (01) in the global prediction table.

Lastly, we set the predictor chooser's state to 10 which weakly chooses the global predictor at start for every row in the counter table.

### *F. Storage analysis*

In this project, we have $64K + 256$ bits to store required informtion in the proposed branch predictor. This is larger than our baseline which only allow $16K$ bits to store information. To utilize the larger storage, we increase the number of bits compared to our baseline.

Our GShare baseline use 13 bits to store the global history. While our tournament predictor utilize 9, 10 and 10 bits to store the global history, the local history and the index for the local history table respectively. These are due to 16K bits budget limitation of the baseline.

In our approach, we will sepearate half of storage for each predictor. Thus, we used 14 global history bits for the global predictor. The rest of the storage is used for local predictor with 11 local history bits and 11 address bits. With this information, we can calculate the total storage used as following.

With 11 bits for both local history and indexing bits, our local predictor used $2^{11} \times 11$ bits for the local history table. Moreover, this means that $2^{11} \times 2$ bits are used for storing the prediciton table. Thus, total used bits are 26624 bits.

For the global predictor, we used 14 global history bits. Thus, the size of the prediction table is $2^14 \times 2 = 32768$ bits which equals to the total storage for this predictor.

In addition to the predictor, we also have the chooser. We used the same index as the local predictor to select the chooser. Therefore, the size of the chooser is $2^11 \times 2 = 4096$ bits.

From these information, the total size of the proposed predictor is $26624 + 14 + 32768 + 4096 = 63502$ bits which still fit to our budget.

## V. Experimental Setup

In the project, we experiment the performance of our predictor compared to the given baselines. Although, baselines have 16K bits limitation. We extended the this limitation by exploring the GShare with 15 bits and the tournament predictor with 11 global history bits, 12 local history bits and 12 program counter index bits. This help show how our predictor preform if our baselines have the same storage budget.

In the testing phase, we have 3 different groups of traces including fp, int and mm. Each trace group have 2 differnt traces, so there are 6 traces in total. We will run each predictor with each trace and measure the misprediction rate. A predictor with a lower misprediction rate means that the performance of it is better.

## VI. Observation

After the experiment was performed and the misprediction rate was collected, we look further into the result we got.

First, we will look at the performance of our baselines. We will represent the GShare with $x$ global history bits as GShare:$x$, and the tournament predictor with $x$ global hisotyr bits, $y$ local history bits and $z$ address index bits as tournament:$x$:$y$:$z$. The performance of baselines is shown in Table I.

TABLE I
Misprediction rate for our baseline predictors with each trace

| Predictor | fp1 | fp2 | int1 | int2 | mm1 | mm2 |
|---|---|---|---|---|---|---|
| GShare:13 (baseline) | 0.825 | 1.678 | 13.839 | 0.420 | 6.696 | 10.138 |
| GShare:15 | 0.827 | 0.985 | 11.220 | 0.364 | 4.441 | 8.039 |
| Tournament :9:10:10 (baseline) | 0.991 | 3.246 | 12.622 | 0.426 | 2.581 | 8.483 |
| Tournament :11:12:12 | 0.990 | 0.410 | 10.171 | 0.333 | 1.072 | 6.967 |

Regarding the Table I, we can see that GShare baseline is better than the tournament baseline for fp trace group. In the other hand, the tournament baseline is better if we consider mm trace group. Both predictors performed great on int trace group. Moreover, although these predictors are extended to larger storage budget, none of these perform better than both original baselines at every traces. The tournament:11:12:12 might look promising, as it can beat 11 out of 12 baseline traces. However, it didn't pass the requirement of the project. That's why we came up with our proposed method. Especially for trace fp1, we believe that using xor result between the program counter and the global history register for the global predictor is the key to beat this trace.

After we implemented our proposed method, we get the result as shown in the Table II. Misprediction rates of our predictor is lower than both baselines for all 6 traces (total 12 traces win). Our predictors is also better than tournament:11:12:12 for 4 out of 6 traces. Moreover, it outperformed GShare:15 for all traces. From these result, this means that our predictor can perform very promising prediction for both local and global branch correlation compared to other methods.

TABLE II
MISPREDICTION RATE FOR OUR METHOD WITH EACH TRACE

| Trace | Misprediction Rate |
|---|---|
| fp1 | 0.811 |
| fp2 | 0.244 |
| int1 | 10.294 |
| int2 | 0.281 |
| mm1 | 1.880 |
| mm2 | 6.550 |

## VII. RESULTS

From previous section, our predictor outperformed both baselines predictor for all traces. It showed that it can solve indexing collision problem which we encountered in fp1 trace which is a problem in the tournament predictor. Moreover, the misprediction rate of our predictor showed that it can handle both local and global branch correlations. The summary of the result for the proposed method and baselines are shown in Table III.

TABLE III
FINAL MISPREDICTION RATE RESULTS FOR BASELINES AND OUR METHOD FOR EACH TRACE.

| Predictor | fp1 | fp2 | int1 | int2 | mm1 | mm2 |
|---|---|---|---|---|---|---|
| GShare:13 | 0.825 | 1.678 | 13.839 | 0.420 | 6.696 | 10.138 |
| Tournament<br>:9:10:10 | 0.991 | 3.246 | 12.622 | 0.426 | 2.581 | 8.483 |
| **Our predictor** | **0.811** | **0.244** | **10.294** | **0.281** | **1.880** | **6.550** |

## VIII. CONCLUSION

In this work, we proposed a branch predictor method which is based on Alpha 21264's tournament predictor. We update index hashing for the global predictor to use xor result between the program counter and the global history register as in GShare to solve index collision problem. We also changed the index of the predictor chooser to use the program counter. The predictor was config to use 14 global history bits, 11 local history bits and 11 address index bits. The total storage used is 63502 bits which fit in our requirement. The proposed predictor was benchmarked with 6 different address traces, and it is compared to GShare with 13 global history bits and the tournament predictor with 9 global history bits, 10 local history bits and 10 address index bits. The result showed that our predictor outperformed the requirement baselines by great amount of margin for all address traces.

papers [1]–[3], you would say something like: "While the authors of [1]–[3] did X, Y, and Z, this paper additionally does W, and is therefore much better." Do NOT omit or anonymize references for blind review. There is one exception to this for your own prior work that appeared in IEEE CAL, arXiv, workshops without archived proceedings, etc. as discussed later in this document.

**References:** There is no length limit for references. *Each reference must explicitly list all authors of the paper. Papers not meeting this requirement will be rejected.* Since there is no length limit for the number of pages used for references, there is no need to save space here.

'Service' collaborations, such as co-authoring a report for a professional organization, serving on a program committee, or co-presenting tutorials, do not themselves create a conflict of interest. Co-authoring a paper that is a compendium of various projects with no true collaboration among the projects does not constitute a conflict among the authors of the different projects. On the other hand, there may be others not covered by the above with whom you believe a COI exists, for example, an ongoing collaboration which has not yet resulted in the creation of a paper or proposal. Please report such COIs; however, you may be asked to justify them. Please be reasonable. For example, you cannot declare a COI with a reviewer just because that reviewer works on topics similar to or related to those in your paper. The program chair may contact co-authors to explain a COI whose origin is unclear.

Most reviews will be solicited among the members of the PC and the ERC, but other members from the community may also write reviews. Please declare all your conflicts (not just restricted to the PC and ERC) on the submission form. When in doubt, contact the program chair.

### A. Concurrent Submissions and Workshops

By submitting a manuscript to ISCA 2020, the authors guarantee that the manuscript has not been previously published or accepted for publication in a substantially similar form in any conference, journal, or the archived proceedings of a workshop (e.g., in the ACM/IEEE digital libraries) — see exceptions below. The authors also guarantee that no paper that contains significant overlap with the contributions of the submitted paper will be under review for any other conference or journal or an archived proceedings of a workshop during the ISCA 2020 review period. Violation of any of these conditions will lead to rejection.

The only exceptions to the above rules are for the authors' own papers in (1) workshops without archived proceedings such as in the ACM/IEEE digital libraries (or where the authors chose not to have their paper appear in the archived proceedings), or (2) venues such as IEEE CAL or arXiv where there is an explicit policy that such publication does not preclude longer conference submissions. In all such cases, the submitted manuscript may ignore the above work to preserve author anonymity. This information must, however, be provided on the submission form — the program chair will make this information available to reviewers if it becomes necessary to ensure a fair review. As always, if you are in doubt, it is best to contact program chairs.

Finally, the ACM/IEEE Plagiarism Policies[1] cover a range of ethical issues concerning the misrepresentation of other works or one's own work.

## Acknowledgements

This work is done alone so I don't have any teammates.

## References

[1] F. Lastname1 and F. Lastname2, "A very nice paper to cite," in *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture*, 2016.

[2] F. Lastname1, F. Lastname2, and F. Lastname3, "Another very nice paper to cite," in *Proceedings of the 48th Annual IEEE/ACM International Symposium on Microarchitecture*, 2015.

[3] F. Lastname1, F. Lastname2, F. Lastname3, F. Lastname4, F. Lastname5, F. Lastname6, F. Lastname7, F. Lastname8, F. Lastname9, F. Lastname10, F. Lastname11, and F. Lastname12, "Yet another very nice paper to cite, with many author names all spelled out," in *Proceedings of the 38th Annual International Symposium on Computer Architecture*, 2011.

[1] http://www.acm.org/publications/policies/plagiarism_policy https://www.ieee.org/publications_standards/publications/rights/plagiarism_FAQ.html