Eng.& Tech. Journal, Vol.30, N0.6, 2012

A Hybrid Neural Based Dynamic Branch Prediction Unit

Gheni A. Ali
AL-Rafidain University College/Baghdad
Email: rafidain_gheni@yahoo.com

Received on: 7/8/2011 & Accepted on: 2/2/2012

ABSTRACT

Modern high performance processor architectures have come to depend upon highly pipelined operation in order to achieve improvements in operating speed. As a result, the cost associated with flushing the pipeline and refilling it when a branch instruction is mis-predicted can significantly impact processor performance. Many schemes, from the extremely simple to the highly complex, have been proposed to improve branch prediction accuracy. Conventional two-level branch predictors predict the outcome of a branch either based on the(local branch history) information, comprising the previous outcomes of a single branch (intra-branch correlation), or based on the (global branch history) information, comprising the previous outcomes of all branches (inter-branch correlation). The misprediction rates for these predictors are very high when they predict branch instructions with hybrid correlations. In this paper we suggest a hybrid perceptron based predictor which employs up to 31-bits of both local and global branch history information to minimize the misprediction rates. The software written for simulation and testing shows that the suggested hybrid predictor achieves a high accuracy. Our results shows that the best response of the predictor is obtained on history length of 16-bits.

Keywords: Branch Prediction, Advanced Processor Architecture, Neural Networks, Pipelining

وحدة ديناميكية هجينة للتنبؤ بالتفرعات باستخدام الشبكات العصبية

الخلاصة

تعتمد معماريات المعالجات الحديثة وبدرجة كبيرة على ما يعرف بالمعالجة التدفقية (Pipelining) من أجل تحسين سرعها التشغيلية. لذلك فإن التكلفة التي تنتج من تفريغ خط المعالجة التدفقية وإعادة ملئه عند حدوث خطأ في التنبؤ بتنفيذ إيعاز تفرع معين سيؤثر بدرجة كبيرة على أداء المعالج. هناك العديد من تقنيات التنبؤ الديناميكي بعمليات التفرع والتي طورت لتحسين دقة التنبؤ، بعض هذه التقنيات بسيطة جدا وبعضها غاية في التعقيد. لكن عموما فإن وحدات التنبؤ التقليدية ذات المستويين تتنبأ بنتيجة إيعاز التفرع أما بناءا على معلومات تاريخية محلية (local branch history) (معلومات عن حالات التنفيذ الفعلية السابقة لنفس الإيعاز الذي يراد التنبؤ بعملية تنفيذه حاليا) وهو ما يعرف بالترابط الداخلي (intra-branch correlation)، أو أن يتم التنبؤ بنتيجة تفرع معين استنادا على معلومات شاملة (global branch history) عن حالات التنفيذ لجميع أوامر التفرع المنفذة مؤخرا. كل الطرق السابقة تقدم دقة تنبؤ واطئة

PDF created with pdfFactory Pro trial version www.pdffactory.com

بإيعازات التفرع التي يعتمد تنفيذها على حالات التنفيذ السابقة للأمر نفسه ولأوامر أخرى وهذا ما يعرف بالترابط الهجين (hybrid correlation). هذا البحث يقدم طريقة هجينة مقترحة للتنبؤ بأنواع التفرعات تستخدم ابسط أنواع الشبكات العصبية وهو (Perceptron). الطريقة التي يقدمها البحث تستخدم لغاية 31 بت من المعلومات الهجينة (محلية وشاملة عن تاريخ تنفيذ إيعاز التفرع) لمراعاة خصوصية الترابط الهجين من أجل تحسين دقة التنبؤ. تم بناء نظام برمجي لمحاكاة واختبار وحدة التنبؤ المقترحة والتي حققت دقة تنبؤ عالية. النتائج التي توصلت لها الطريقة تبين أن انسب استجابة تتحقق عند 16 بت من المعلومات الهجينة.

INTRODUCTION

Most commercial processors are implemented on pipeline and superscalar architecture. In superscalar architecture, branch instructions may reduce the parallelism because the branch direction or the target address during the instruction fetch can not be known [1]. A conditional branch instruction introduces the added hazard caused by the dependency of the branch condition on the result of the preceding instruction. The decision to branch cannot be made until the execution of that instruction has been completed [2].

Branch instructions occur frequently, in fact, they represent about 20 percent of the dynamic instruction count of most programs (The dynamic count is the number of instruction executions, taking into account the fact that some program instructions are executed many times because of loops). Because of the branch penalty, this large percentage would reduce the gain in performance expected from the pipelining. Fortunately, branch instructions can be handled in several ways to reduce their negative impact on the rate of execution of instructions. One of these ways is the branch prediction [2].

Branch prediction represents the process of correctly predicting the branch's direction and target address before it is actually executed. High accuracy branch prediction is increasingly important in today's superscalar and deep pipeline processor architecture [3]. Statistically was proven that conditional branches are executed about every 7 to 8 instructions at average. Current wide issue architectures can execute ***four*** or more ***independents*** instructions per clock cycle. So, a branch instruction is likely to be executed every ***two*** clock cycles or less. This means that branch prediction is crucial for processor performance [4]. According to the time and way the prediction is resolved, Smith [5] classify branch prediction into two categories: Static and Dynamic. Static branch prediction is simpler and depends mainly on program structure. For example, a branch instruction at the end of the loop causes a branch to the start of the loop for every pass through the loop except the last one. Hence, it is advantageous to assume that this branch will be taken and to have the instruction fetch unit start to fetch instructions at the branch target address. On the other hand, for branch instructions on the beginning of a program loop, it is advantageous to assume that the branch will not be taken. The strategy described above is called FTBNT (Forward Taken Backward Not Taken).

PDF created with pdfFactory Pro trial version www.pdffactory.com

A backwards branch is a branch instruction that has a target with a lower address (i.e. one that comes earlier in the program) [6].

Generally, Static branch prediction algorithms tend to be very simple, and by definition do not incorporate any feedback from the run-time environment. By not paying any attention to the dynamic run-time behavior of a program, the branch prediction is incapable of adapting to changes in branch prediction patterns. The advantage of static branch prediction techniques is that they are very simple to implement, and do require very little hardware resources. Static branch prediction algorithms are of less interest in the context of future generation [7]. A dynamic algorithm keeps a record of previous branch behavior, allowing it to improve its predictions over time. A simple scheme, published by James Smith [5], maintains a single history bit for each branch. When a branch is encountered, it is predicted to go the same way it did the previous time, as indicated by the bit. This technique which used by Digital's Alpha, AMD's K5, and other processors, can push accuracy to 80% [8]. Processors such as Pentium store the history bits in a separate Branch History Table (BHT), assigning one entry per branch to achieve improved accuracy. Alternatively, similar accuracy is achieved with fewer entries. The BHT, however, must maintain its own set of tags, greatly increasing the amount of storage required.

Given the overhead of tag storage, most processors with a separate BHT store two bits of history per entry instead of just one bit. In this method, also elucidated by Smith [5], the two bits can be thought of as a saturating counter that is incremented when the branch is taken and decremented when it is not; the most-significant bit is used to predict future occurrences. Another way to look at this implementation is as a state machine, which is depicted in Fig. 1. In two bit Smith algorithm (recent literature has often referred to it as "bimodal" prediction), the two history bits implement a state machine with four possible states: strongly taken (ST), weakly taken (WT), weakly not taken (WNT), and strongly not taken (SNT). In ST and WT, future branches are predicted taken; in WNT and SNT, branches are predicted not taken. The advantage of bimodal method is that a single unusual iteration will not change the predicted direction. For example, if a branch has been taken many times in succession, the state machine will be in the Strongly Taken state (3). If the branch is then not taken, the history bits will indicate Weakly Taken but still predict the next iteration as taken. Only if the branch is not taken two or more times consecutively will the prediction change to not taken. This hysteresis effect can boost prediction accuracy to 85% , depending on the size and type of history table that is used [8].

TWO LEVEL PREDICTIONS

Bimodal prediction can be improved in two ways, both of which explicitly track prior branch outcomes and were introduced by Yeh and Patt [9]. Local-history prediction maintains a table of per-branch histories. Instead of tracking each branch's predominant direction, this Branch History Table (BHT) tracks explicit history in order to detect patterns. For example, a local history can detect patterns like TNTN… that confound simple saturating counters. The predictor still keeps a PHT (Pattern History Table) of two-bit counters, but these are now indexed using

PDF created with pdfFactory Pro trial version www.pdffactory.com

the local history pattern, and the counters now learn outcomes for each history pattern. A schematic of a local history predictor appears in fig. 2.

All local branch predictors, predict the outcomes of a branch based on the local branch history, i.e. the previous outcomes of the branch itself. This kind of correlation between the results of a single branch is called intra-branch correlation. The Intra-branch correlation often arises from loop branches with a regular number of iterations and branches with periodic outcome patterns. Consider the example in Fig.3: the While-Do loops in the code fragment will be translated to conditional branches, denoted as branches E and F, after compilation. Branches E and F will be taken nine times and then untaken once repeatedly. Per-address two-level predictors keep a dedicated BHR for each branch to record its previous outcomes. If a branch has periodic outcomes and the length of the BHR is long enough to capture the whole periodic pattern, per-address two-level predictors are able to predict the result of the branch perfectly. In real programs many branches exhibit intra-branch correlation, which is why per-address two-level (local) predictors work well .

The table of per-branch histories (Branch History Table BHT) can be replaced with a single, global shift register, the Global Branch History Register (GBHR). All branches shift their outcomes into this register, which typically is 10 -15 bits wide. This may seem a strange thing to do, but global history prediction allows branches to easily see the behavior of other recent branches [9]. A typical global history predictor appears in Fig. 4. This figure. also shows the inclusion of some address bits in the index for the table of two-bit counters (the pattern history table PHT). The preceding discussion has assumed each branch has a unique entry in these tables, but these are hardware structures and necessarily of finite size.

Two branches may therefore share the same entry, either because the table is not sufficiently large or because the two branches share the same prior history. This may be harmless, and sometimes even helps when it accidently permits related branches to communicate additional information among each other. But if often results in destructive interference as the branches overwrite each others' state. This aliasing can be alleviated in both global and local history predictors by combining the history bits with some bits from the branch's address. One simple way to do this, proposed by McFarling [10] is to XOR the two patterns together, creating a gshare predictor. Now branches that share the same history are usually distinguished by their different addresses. The code example shown in Fig. 5, summarize the second type of correlation; the inter-branch correlation. There are four if-statements in the code fragment, and they will be translated to conditional branch instructions after compilation. In this example, branches C and D are correlated with branches A and B. When the results of branches A and B are determined, the results of branches C and D are also determined. A global two-level branch predictor is able to record the various outcomes of branches C and D corresponding to different outcome combinations of branches A and B. When a specific outcome combination of branches A and B happens again, the branch predictor can predict the results of branches C and D by looking up the pattern history table. In real programs there exists a large amount of inter-branch

PDF created with pdfFactory Pro trial version www.pdffactory.com

correlation, which is the reason why global two-level branch predictors can work [12].

Some branches, like that shown in the example of fig. 6, are not predictable based on merely the global branch history or merely the local branch history. This example contains three if-statements, each of which will be translated to a conditional branch after compilation. We shall use the notation C1 to denote the condition (I mod 35 = 0) , and the notation C2 to denote the condition (I mod 3<> 0). In this example, branch Z has partial correlation with branches X and Y. That is, if the results of branches X and Y are both known, the condition C1 can be determined. However, since the outcome of branch Z also depends on condition C2, it is not sufficient to derive the outcome of branch Z from the outcomes of branches X and Y. In this example, global two-level branch predictors cannot predict the outcome of branch Z exactly due to lacking the information of C2. Similarly, local two level predictors cannot predict the outcome of branch Z exactly because the BHR fails to capture the whole periodic outcome pattern of branch Z unless the BHR has a nonrealistic length of more than 105 bits. M.-C. Chang and Y.-W. Chou [12] proposed a branch predictor, called LGshare, which exploits both of the global branch history and the local branch history simultaneously to enhance branch prediction accuracy. The LGshare predictor has an n-bit global BHR to record the recent n outcomes of all branches and an m-bit local BHR for each branch to record the m recent outcomes of the branch. Every time when the PHT is to be accessed, the m-bit history in local BHR and the n-bit history in global BHR are concatenated to form the hybrid branch history, and then the hybrid branch history is XORed with the branch address to form the index to PHT.

BRANCH PREDICTION USING NEURAL NETWORKS

The first perceptron based dynamic branch prediction was proposed by Jimenez and Lin [11]. Fig.(7) shows a graphical model of a sample branch predicting perceptron. The input values x_1 through x_n are prior branch outcomes coming from the global branch history register. These are bipolar; each x_i is either 1, meaning the branch was taken, or –1, in the case which the branch was not taken. Weights w_1 through w_n are weights associated with their respective input, the larger the absolute value of w_i, the higher degree of correlation of x_i has with the output. These values come from a table of weights, indexed by the branch address. The output, y, is computed as the dot product of these weighted input vectors. According to the following equation:

$$y = bias_{_w} + \sum_{i=1}^{n} x_i w_i \qquad \text{…………(1)}$$

Another neural based branch predictor is given by P.B. Osofisan, and O.A. Afunlehin, [13]. In their work they used two types of neural networks; Back propagation and Learning Vector Quantization (LVQ) nets. These methods which are formerly used by others, includes some limitations that makes them not attractive solutions to implement efficient predictors. Because of its

PDF created with pdfFactory Pro trial version www.pdffactory.com

implementation complexity, there is no way to implement back-propagation in hardware such that a prediction can be produced in just a few cycles. While LVQ does not lend itself well to high-speed implementation because it performs complex computations involving floating point numbers [14].

THE PROPOSED PREDICTOR

The main purpose of our work is applying the most simple neural network (perceptron) in the implementation of LGshare predictor described in section 2 above. Fig.(8). Shows the structure of the proposed predictor.

PREDICTOR STRUCTURE

The suggested structure includes a branch target buffer which contains the per – branch history for number of branch instructions. The size of BTB depends on the number of branch instructions that it can contains their history and the number of history bits for each branch. Each branch instruction associates with an entry in the perceptron table, this table is a two dimensional array, each entry in the row represent the weight, which is an integer whose value dictates how strongly the current branch correlates with a corresponding entry in the LBHR, these weights are updated dynamically according to the training rules which will be described later. With each branch instruction the BTB is accessed, in the case of hit, its history is loaded into the LBHR. Some bits of LBHR is concatenated with a portion of GBHR to formulate a special register called HBHR which contains some history about this branch and the most recently occurrence of some other branches that may affect the execution of the present one. The suggested contribution of local history and global history in the formulation of HBHR depends on the history length of HBHR. Local bits always will take the RHS part of the HBHR (the least significant bits) Table (1). Summarize this contribution. The history bits are stored in binary (0 for *not taken*, and 1 for *taken*), but in the time of processing (prediction and training) they are converted into bipolar; either 1, in the case of taken, or -1 , when the branch was not taken. XORing branch address with its local history is used to alleviate the aliasing that may be occurs if two different branches have same history. Actually, the formulation of HBHR is a bit manipulation operation between GBHR and LBHR of the branch to be predicted, we use the built-in assembler to perform this operation. Fig. 9, shows a simple assembly code to formulate a HBHR with length of 15-bits (7 global bits, 8 local bits)

PERCEPTRON PREDICTION AND TRAINING

The perceptron, shown in fig.. (7)., like any neural network, must be trained in order to operates properly. The training is done by changing the value of each weight according to the actual branch occurrence. The training (weights updating) is only done in the case of a misprediction or if the output value of the perceptron is less than or equal to a certain value called the threshold. The threshold value depends mainly on the number of history bits [11], and it is calculated by:

$$threshold = 1.93 * history_length + 14 \qquad \dots\dots\dots(2)$$

PDF created with pdfFactory Pro trial version www.pdffactory.com

The perceptron prediction is implemented by applying equation 1. The code shown in fig. (10), shows the implementation of perceptron prediction. The history (local and global) is updated with each branch execution, while weights updating is done under certain conditions fig. (11) shows the code for weight updating:

SIMULATION RESULTS

In order to evaluate the prediction accuracy of the hybrid predictor, a program is written in Turbo Pascal 7 and its associated built in turbo assembler. All of the conditional branches to be tested are gathered by recording their actual outcomes in a special matrix called **Actual**, while their addresses and history (initially assumed to be taken i.e. all history bits are 1's) are recorded in BTB. For each branch, this simulator program gets the branch address and predicts the direction of the branch according to the code shown in fig. 10). It then compares the prediction with the actual outcome to collect the statistics of prediction accuracy.

The prediction accuracy is calculated according to equation (3).

$$Accuracy = \frac{No.\ of\ correctly\ predicted\ branches}{Total\ No.\ of\ tested\ branches} \quad \ldots\ldots (3)$$

While the percentage of misprediction rate is calculated using equation 4.

$$Misprediction\ rate = 1 - Accuracy \quad \ldots..(4)$$

The test program includes different types of branch instructions with various correlation types. The same conditional branches are tested using local, global and suggested hybrid history scheme. Fig. (12). shows the relationship between the history length and misprediction rates for different types of history information. Clearly that the local history based predictor provides the best performance for most of the tested history lengths, while the suggested predictor gives a better response than the global one for the range (2-18) bits while they behaves likely for history lengths of 19 bits and above. Fig.(13). describes the behavior of the three predictors when they process the hybrid correlated branch instructions only. It is clear that the local predictor gives a better response for history lengths (4-10 bits) and there is no significant improvements in its performance beyond history length of 11 bits. While the suggested hybrid predictor provides best performance between history lengths of (11 – 18) bits.

HARDWARE BUDGET

The hardware required to implement the suggested unit depends mainly on history length, number of BTB entries (branches) to be considered and the type of data will be allocated for weights. The weights for the predictor are signed integers. Although many neural networks have floating-point weights, we found that integers are very sufficient, and simplify the design. We find that using integer weights provides the best trade-off between accuracy and hardware budget. Table

PDF created with pdfFactory Pro trial version www.pdffactory.com

(2), shows a summary for a hardware budget for a 32-entries BTB. Note that the registers size is excluded from the budget

CONCLUSIONS

Branch prediction is important in high-performance processors and its importance continues to grow. In the drive for higher execution frequencies, pipelines are lengthened and memory latencies are increased. This increases the cost of branch mispredictions. To alleviate the negative impact of conditional branches, a branch predictor with very high accuracy is essential to a superscalar processor. Conventional two-level branch predictors make predictions either based on global branch history only or based on local branch history only. In this paper a dynamic neural branch predictor is proposed. Different types of branches are considered including that with hybrid correlation. The written program tests the suggested predictor for history lengths of up to 31 bits. Figures 14 and 15 shows that the hybrid predictor provide a high performance especially in predicting the branches with hybrid correlations. There are many parameters effects on the performance of the predictor. The most important ones are: history length, threshold and initial assumptions of the weights and branch history. Our study shows that the best results are obtained when the initial history is always taken (T,T,T,T,….,T) for local branches history and (T, NT, T, NT, T, NT…) for global branches history. Also the results of the study summarized in figures 14 and 15 shows that the best response is obtained at history length of 16 bits.

REFERENCES

[1] Ribas, V. M. Figueiredo and R. Goncalves, ***"Simulating a simple neural network on branch prediction"***, Acta Scientiarum Technology V. 50, no. 2, pp 153-160, 2003.

[2] Hamacher, C. Z. Vranesic and S. Zaky, ***"Computer Organization"***, 5th edition, Mc Graw Hill, 2002.

[3] Steven, G. B. B. Christianson, R. Collins, R. Potter, and F. Steven, ***" A Superscalar Architecture to Exploit Instruction Level Parallelism"***, Microprocessors and Microsystems, Vol.20, No 7, March 1997, pp.391-400.

[4] Sbera, M. ***"Some contributions to static and dynamic branch prediction challenge"***, MSc. Thesis University of L. Blaga, sibiu, Romania, July 2001.

[5] J. E. Smith, ***" A study of branch prediction strategies"***. Annual International Symposium On Computer Architecture, Minneapolis, 1981, pp. 135-148.

[6] G. H. Loh, ***" Microarchitecture for Billion-Transistor VLSI Superscalar Processors"***, Ph. D. Dissertation, Yale University,2002.

[7] Kaeli, D. and P. Chung Yew, ***"Speculative Execution in High Performance Computer Architectures",*** Taylor & Francis Group, 2005.

[8] *Gwennap, L.* ***"New Algorithm Improves Branch Prediction, Better Accuracy Required for Highly Superscalar Designs",*** *Micro Design Resources, Vol. 9, No. 4, March 27, 1995*

[9] *Yeh, T.-Y. and Patt, Y. N.* ***"Two-level adaptive training branch prediction"***, *, in Proc. 24th Ann. Int. Symp.on Microarchitecture, pp. 51–61, November 1991.*

PDF created with pdfFactory Pro trial version www.pdffactory.com

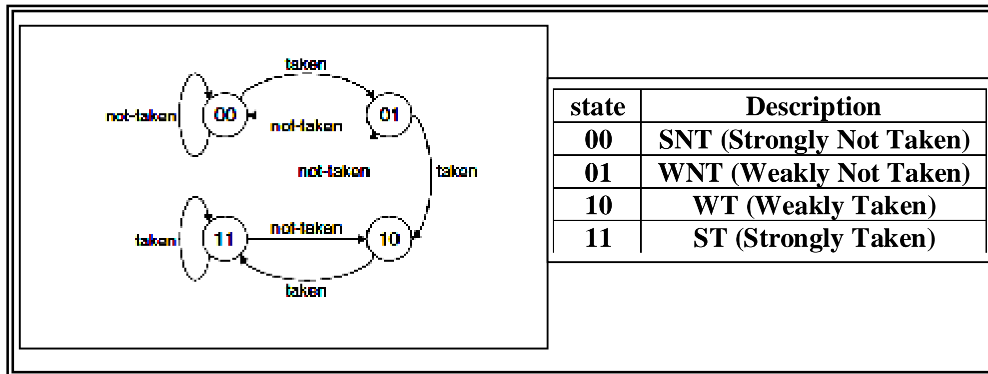
[10] *McFarling.S.* ***"Combining branch predictors"*** *. Tech. Note TN-36, DEC WRL, June 1993.*

[11] *Jimenez, D. A. and C. Lin,* ***"Dynamic Branch Prediction with Perceptrons"***. Proceedings of the 7th International Symposium on High-Performance Computer Architecture, Jan. 2001.

[12] *Chang, M.-C. and Y.-W. Chou.* ***" Branch prediction using both global and local branch history information",*** IEE Proc.-Comput. Digit. Tech., Vol. 149, No. 2, March 2002.

[13] Osofisan, P.B. and O.A. Afunlehin, **"Application of Neural Network to Improve Dynamic Branch Prediction of Superscalar Microprocessors",** The Pacific Journal of Science and Technology, Volume 8. Number 1. May 2007 (Spring).

[14] Jimenez and C. Lin**,** D. A. **" Neural Methods for Dynamic Branch Prediction",** ACM Transactions on Computer Systems, Vol. 20, No. 4, November 2002, Pages 369–397.



state	Description
00	**SNT (Strongly Not Taken)**
01	**WNT (Weakly Not Taken)**
10	**WT (Weakly Taken)**
11	**ST (Strongly Taken)**

Figure (1) a Simple State Machine for Branch Prediction (Two Bit Smith Algorithm)

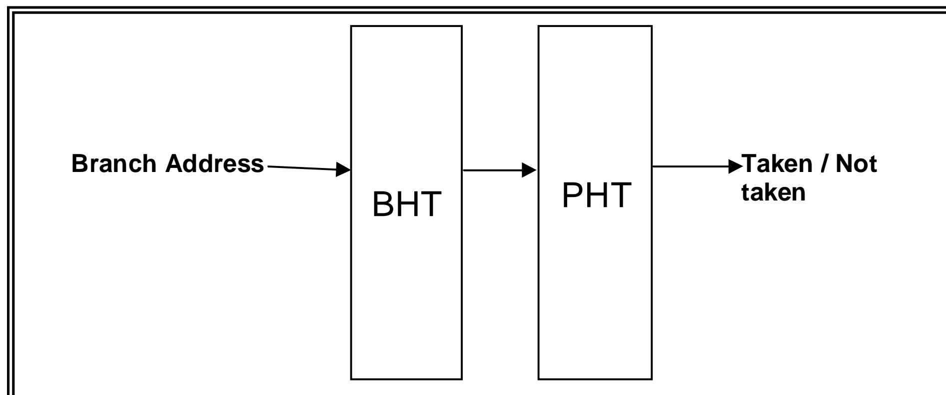


Figure (2) a local-history-based two-level predictor

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
i:=1؛
while(i<10) do{Branch E}
begin
j:=1؛
while (j<10) do {branch F}
begin
writeln(i,' * ',j,' = ',i*j);
j:=j+1؛ end؛
i:=i+1؛ end؛
```

Figure (3) a code for an example of intra-branch correlation

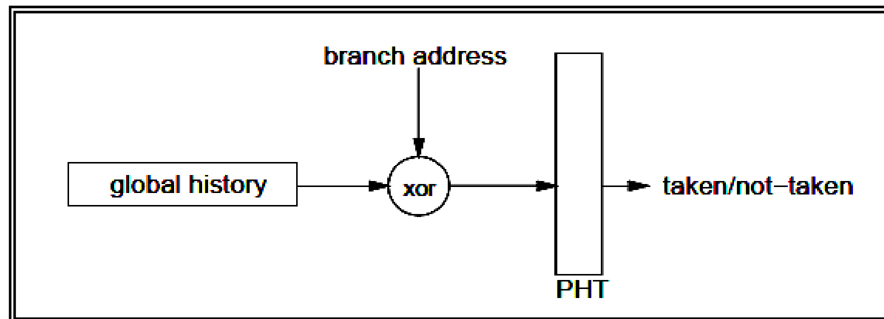


Figure (4) a global-history-based two-level predictor.

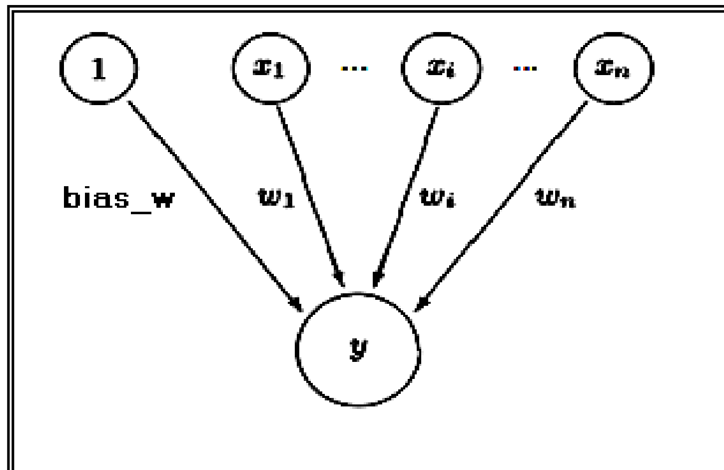
```
if ( x < y) then flag1:= 1; { Branch A}
if ( x < z) then flag2:= 1; { Branch B }
if ( x < y) or ( x < z ) then  { Branch C }
writeln (' x is not larger ');
if (flag1 = 1) and (flag2 = 1) then { Branch D }
writeln (' x is smallest ');
```

Figure (5) a code for an example of inter-branch correlation

PDF created with pdfFactory Pro trial version www.pdffactory.com

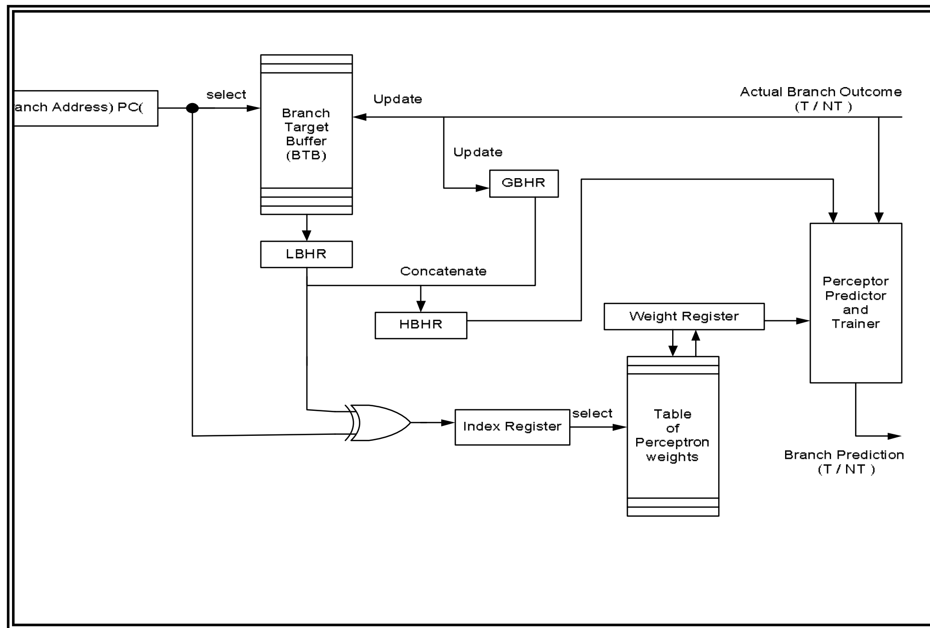
```
i : integer؛
begin
i:=1؛
while (i < 1000 ) do begin
if (i mod 5 = 0 ) then   { branch X }
writeln ( ' 5 divides   ',i )
if (i mod 7 = 0 ) then   { branch Y  }
writeln ( ' 7 divides   ', i )؛
if ( i mod 35 = 0) and ( i mod 3 <> 0) then {branch Z  }
begin
writeln (' 35 divides   ',i );
writeln (' 3 does not divides   ', i);
end؛
i:= i + 1؛
end؛
end;
```

Figure (6) A code for an example of hybrid correlation



Figure(7) Perceptron Model

PDF created with pdfFactory Pro trial version www.pdffactory.com



GBHR (Global Branch History Register), LBHR (Local Branch History Register)
HBHR (Hybrid Branch History Register)., T / NT (Taken / Not Taken)

Figure 8. The structure of the proposed perceptron based Hybrid predictor

Table1. Contribution of Local and Global Histories in the formulation of HBHR

HBHR LENGTH (BITS)	GLOBAL HISTORY (BITS)	LOCAL HISTORY (BITS)	HBHR LENGTH (BITS)	GLOBAL HISTORY (BITS)	LOCAL HISTORY (BITS)
2	1	1	17	8	9
3	1	2	18	9	9
4	2	2	19	9	10
5	2	3	20	10	10
6	3	3	21	10	11
7	3	4	22	11	11
8	4	4	23	11	12
9	4	5	24	12	12
10	5	5	25	12	13
11	5	6	26	13	13
12	6	6	27	13	14
13	6	7	28	14	14
14	7	7	29	14	15
15	7	8	30	15	15
16	8	8	31	15	16

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
procedure concat15; assembler؛
label b1؛
label b2؛
asm
push ax؛
push bx؛
push cx؛
push dx؛
push si؛
push di؛
        mov bx, offset GBHR+2؛
        mov si,[bx؛[
        mov bx, offset GBHR؛
        mov ax,[bx؛[
        mov cl,8؛
b1:     shl si,1؛
        shl ax,1؛
        jnc b2؛
        or si,01H؛
b2:     dec cl؛
        jnz b1؛
        mov bx,offset LBHR؛
        mov dx,[bx؛[
        and dx,00ffH؛
        or ax,dx؛
        mov bx, offset HBHR؛
        mov [bx], ax؛
        mov bx, offset HBHR+2؛
        mov [bx],si؛
pop di؛
pop si؛
pop dx؛
pop cx؛
pop bx؛
pop ax؛
end؛
```

Figure (9) Assembly language procedure for formulation of 15-bits HB

PDF created with pdfFactory Pro trial version www.pdffactory.com

```
sum := 0;
For i:= 1 to HBHR_length do
sum := sum + Weight[i] * HBHR[i];
sum := sum + bias_w[i];
if sum >= 0 then prediction := +1 { Prediction is Taken}
else prediction := -1; {prediction is Not Taken}
```

Figure (10) A simple code for implementation of Perceptron Prediction

```
if (prediction <> actual_BranchOutcome) and (abs(sum) <= threshold) then
begin
for i:= 1 to HBHR_length do begin
if HBHR[i] = actual_BranchOutcome then
weight[i]:= weight[i] + 1 else
weight[i]:= weight[i] – 1
end;
if actual_BranchOutcome =  1 then
bias_w[i]:= bias_w[i]+1 else
bias_[i]:= bias_w[i] -1;
end;
```

Figure (11) A simple code for implementation of Perceptron's weight updating

PDF created with pdfFactory Pro trial version www.pdffactory.com

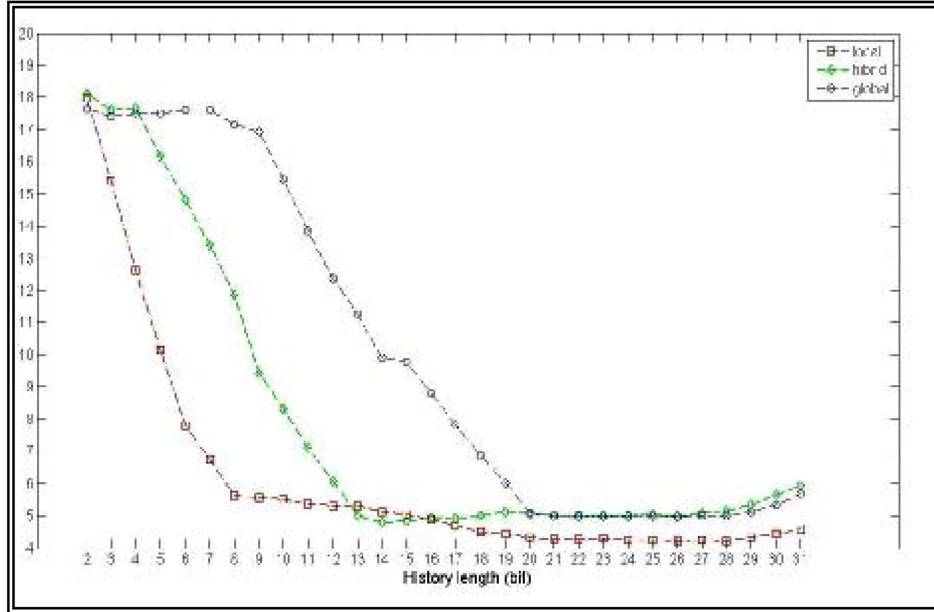


Figure (12) Performance comparisons of three branch predictors with different history lengths in processing different types of correlated branches

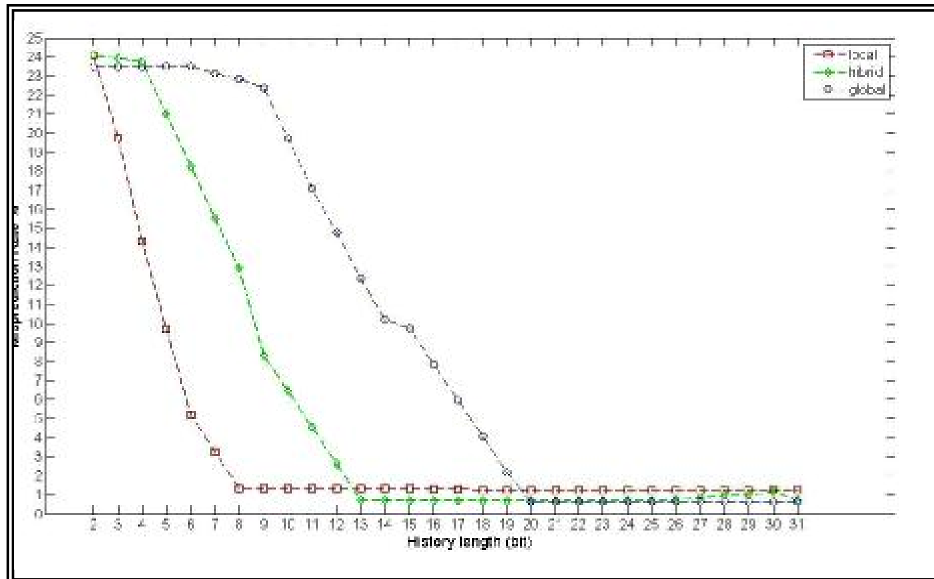


Figure (13) Performance comparisons of three branch predictors with different history lengths in processing branch instructions with hybrid correlations

PDF created with pdfFactory Pro trial version www.pdffactory.com

Table (2) A summary for the hardware budget for the suggested predictor assuming a BTB with 32 entries only

HISTORY LENGTH (BITS)	BTB SIZE (BITS)	PERCEPTRON TABLE SIZE (BITS)	TOTAL HARDWARE BUDGET (BITS)
2	1056	1536	2592
3	1088	2048	3136
4	1088	2560	3648
5	1120	3072	4192
6	1120	3584	4668
7	1152	4069	5221
8	1152	4608	5760
9	1184	5120	6304
10	1184	5632	6816
11	1216	6144	7360
12	1216	6656	7872
13	1248	7168	8416
14	1248	7680	8928
15	1280	8192	9472
16	1280	8704	9984
17	1312	9216	10528
18	1312	9782	11094
19	1344	10240	11584
20	1344	10752	12096
21	1376	11264	12640
22	1376	11776	13152
23	1408	12288	13696
24	1408	12800	14208
25	1440	13312	14752
26	1440	13824	15264
27	1472	14336	15808
28	1472	14848	16320
29	1504	15360	16864
30	1504	15872	17376
31	1536	16384	17920

PDF created with pdfFactory Pro trial version www.pdffactory.com