LSU EE 4720 **Homework 5** Solution **Due: 28 April 2006**

Note: For some sample problems with predictors see the final exam solutions.

Problem 1: The routine `samples` in the code below is called many times. Consider the execution of the code on three systems, each system using one of the branch predictors below.

All predictors use a 2^{14}-entry branch history table (BHT). (The global predictor does not need its BHT for predicting branch direction.) The three predictors are:

- System B: bimodal
- System G: global, history length 10. (Accuracy can be approximated.)
- System L: local, history length 10.

```
void samples(int& x, int& y, char **string_array )
{
  // Loop 5-xor
  for( int i = 0; i < 5; i++ )
    x = x ^ i;

  // Loop 5-len
  for( int i = 0; i < 5; i++ )
    if( strlen( string_array[i] ) < 20 )
      return; // Never executes.             <- Important.

  // Loop 100-xor
  for( int i = 0; i < 100; i++ )
    y = y ^ i;
}
```

(a) Determine the amount of memory (in bits) needed to implement each predictor.

Bimodal: The bimodal predictor just uses the BHT, each entry is two bits. Bimodal predictor size $2^{14} \times 2$ bits.

Global: The global predictor uses only a pattern history table, the number of entries is 2^h, where h is the history length, 10 in this case; each table entry is 2 bits. Global predictor size $2^{10} \times 2$ bits.

Local: The local predictor uses both a BHT and PHT. The BHT stores the local history, which is the outcome of the last 10 branches, so each entry is 10 bits. The PHT is indexed using this 10-bit history so there are 2^{10} entries, each entry of this table is 2 bits. Local predictor size $2^{14} \times 10 + 2^{10} \times 2$ bits.

Continued on next page.

(*b*) For each loop in `samples` determine the accuracy of the loop branch (the one that tests the value of `i`) after warmup on each system. The accuracy for the global predictor can be approximated, the others must be determined exactly.

Bimodal: Once warmed up the bimodal predictor will only mispredict the last loop iteration. Loops 5-xor and 5-len are 5 iterations and 100-xor is 100 iterations. Bimodal accuracy on 5-xor and 5-len is $\frac{4}{5} = 0.8$.

Bimodal accuracy on 100-xor is $\frac{99}{100} = 0.99$.

A solution for 5-xor is worked out in detail below. Accuracy is computed over a repeating region, in this case outcome # 5 to 10. At these outcomes the loop has just started and in both cases the counter value is 2.

```
Loop 5-xor - Bimodal
 Outcome #          0  1  2  3  4  5  6  7  8  9  10  : Not part of predictor
 Branch Outcomes    T  T  T  T  N  T  T  T  T  N  T...: N, Not taken; T, Taken
 2-bit Counter    0  1  2  3  3  2  3  3  3  3  2     : BHT entry
 Prediction        N  N  T  T  T  T  T  T  T  T       : T if counter > 1
 Mispredicted       X  X        X              X      : X means misprediction
 Accuracy: 4 correct predictions / 5 predictions
```

Local: The local predictor can predict short loops perfectly so long as the number of iterations is not larger than the history length. That is the case for 5-xor and 5-len. Local predictor accuracy on 5-xor and 5-len is 1.0.

For loop 100-xor the local predictor will mispredict the last iteration. Local predictor accuracy on 100-xor is 0.99.

Global: The global predictor can also predict short loops as long as the outcome of the first loop branch is present when predicting the last iteration. For loop 5-xor there is no other branch in the loop and so the global history is long enough to distinguish the last iteration from others. Global predictor accuracy on 5-xor is 1.0.

Loop 5-len includes a call to the **strlen** routine, and that most certainly has branches. There is also the branch testing the string length. In the diagram below the 5-len branch outcomes, the **if** statement branch outcomes, and the **strlen** branch outcomes are shown. The global history is a concatenation of these outcomes. When trying to predict outcome 1 the global history will be TTTTTTTTNN (from the loop in the strlen routine), that's the same global history when trying to predict outcomes 2, 3, and 4, so there is no way to distinguish outcome 4 from outcomes 1, 2, and 3. Since most of those are taken the PHT entry will be 2 or 3 and Taken will be predicted every time, the same prediction a bimodal would make. Global predictor accuracy on 5-len is 0.8.

```
Loop 5-len - global
 Outcome #          0       1        2       3        4
 FOR branch:        T       T        T       T        N
 IF branch                N        N       N        N
 strlen branch:      TT..TN   TT..TN  TT..TN   TT..TN
```

(*c*) Why would solving the problem above be impossible, or at least tedious, if the BHT size were $\approx 2^3$ entries?

If the BHT were that small there would be a reasonable chance that more than one of the branches above would occupy the same BHT entry, possibly reducing accuracy. Since the code is given in high-level form there is no way to tell what the branch addresses were. Unless you compiled the code and examined the branch addresses. That would be tedious.

There's `more` *on the next page.*

Problem 2: The code `more`, below, runs on four systems. All predictors use a 2^{14}-entry branch history table (BHT). (The global and gshare predictors do not need their BHT for predicting branch direction.) The predictors are:

- System B: bimodal
- System G: global, history length 10. (Accuracy can be approximated.)
- System X: gshare, history length 10. (Accuracy can be approximated.)
- System L: local, history length 10.

(*a*) In the code below estimate the prediction accuracy of the following predictors on Branch B and Branch C (there is no Branch A) after warmup, assuming that `more` is called many times.

Bimodal: Each branch's outcome is always the same (they are *highly biased*) and the BHT is large enough so that it's unlikely that any branch shares B or C's entry. Bimodal predictor accuracy on B and C is 1.0.

Local: The local history for B after warmup will always be "NNNNNNNNN" and the corresponding PHT entry will have sunk down to zero (if not already reduced to zero by other biased not taken branches); C's local history after warmup will always be "TTTTTTTTTT" and the corresponding PHT entry will be 3. Local predictor accuracy on B and C is 1.0.

Global: The global history used when predicting B consists of the outcomes from the 100-iteration loop above B, that history will be "TTTTTTTTTN". The global history used when predicting C consists of the outcomes from the 100-iteration loop above C; it's a different but similar loop and so the outcomes will be the same "TTTTTTTTTN". Since both B and C are predicted using the same global history they will share a PHT entry, branch B will decrement the entry and branch C will increment it. Assume that no other branch uses that PHT entry. If the entry starts out at 0 or 1 branch B will be predicted with 100% accuracy and C will be predicted with 0% accuracy. If the entry starts out at 3 branch B will be predicted at 0% accuracy and C will be predicted at 100% accuracy. If the entry starts out at 2 both B and C will be predicted at 0% accuracy. Global predictor accuracy on B and C: 1 and 0 or 1 and 0 or 0 and 0.

gshare: In a global predictor the PHT is indexed using the global history register. A gshare predictor is identical to a global predictor except that the PHT is indexed using the bitwise exclusive or of the global history and the branch address. As a result the PHT entries used for predicting branches B and C will be different. (The global histories are still the same but of course the addresses of B and C are different.) Assuming no other branches share these PHT entries, the prediction accuracy will be 100%. gshare accuracy on B and C: 1.0.

(*b*) One of the predictors should have a low prediction accuracy. Why? Avoid a sterile description of the hardware, instead discuss the concept the predictor is based on and why that's not working here.

The global predictor has the low prediction accuracy. See the solution to the previous part for the global and gshare predictors.

```
void more(int& x, int& y, int a, int& b, int& c)
{
  for( int i=0; i<100; i++ ) x = x ^ i;

  if( a < 10 ) b++;  // Branch B, never taken.

  for( int i=0; i<100; i++ ) y = y ^ i;

  if( a >= 10 ) c++; // Branch C, always taken.
}
```