CIS 629 - Take Home Midterm

Introduction
Resolving branch addresses has always been an obstacle in realizing the full potential of pipelining. Several branch prediction schemes have been proposed which attempt to predict the target branch address once the branch instruction has been decoded. As of 2001, the most promising proposal is the Alpha 21264 Tournament Branch Predictor. Below, is a series of studies to investigate the inner workings of the predictor, an analysis of the results of each study, and recommendations to make improvements to the predictor.

Overview of the Predictor
The Alpha 21264 Branch Predictor uses both global and local history to predict branch target addresses. It has 4K entries in its selector table, each of which is a 2-bit counter whose value determines whether the global or local predictor is taken. Entries in the selector table are indexed by the local branch address. The global predictor is also made up of 4K 2-bit entries which are indexed by the global history of the last 12 branches. The local predictor is a two-level predictor where its top level is a table of 1024 10-bit entries where entry i represents the history of the 10 most recent branch outcomes for branch i. The values from this table are then used to index the lower level of the local predictor which has 1K entries consisting of 3-bit counters.

Methodology

Setup
Because only empirical experiments can be performed, observations will be made solely by running several different benchmarks on the Alpha 21264 processor, and analyzing the results.
In order to evaluate the behavior of the predictor, a selected group of benchmarks from the SPECCPU 2000 suite will be used. A list of the benchmarks to be used is listed below.

**Integer**
gzip - data compression
crafty - chess-playing program
gcc - C programming language compiler
vpr - circuit placement and routing program
mcf - combinatorial optimization

**Floating-Point**
wupwise - physics/quantum chromodynamics
mgrd - multi-grid solver: 3D potential field
art - image recognition/neural networks
lucas - number theory/primality testing

It is important that a relatively large number of integer benchmarks be used since branch directions are harder to predict in those types of programs. All of the integer benchmarks chosen have a high number of conditional branches, and exhibit behavior which is hard to predict at runtime. The floating-point benchmarks were chosen randomly to reflect a diverse set of calculations, although loops (which are more predictable) tend to be the type of conditional branch prevalent in these programs.

Each benchmark will run on the Alpha 21264 processor for 100 iterations, and statistics reported will represent the average across all iterations. No command-line parameters will be added to generate optimizations by the compiler. Several pieces of information (listed in more detail below) will be recorded in order to assess the frequency that the global/local predictor is used, and the accuracy of each one when making predictions.

Metrics
The following counters will be used to record measurements for each benchmark during testing.

<table>
<thead>
<tr>
<th>Counter Label</th>
<th>Measurement</th>
<th>Incremented when:</th>
</tr>
</thead>
<tbody>
<tr>
<td>CF</td>
<td>Conditional Frequency - number of condition branches</td>
<td>the ID stage of the pipeline HW detects a conditional branch</td>
</tr>
<tr>
<td>LF</td>
<td>Local Frequency - number of times local predictor is selected</td>
<td>the local level is selected from the predictor</td>
</tr>
<tr>
<td>GF</td>
<td>Global Frequency - number of times global predictor is selected</td>
<td>the global level is selected from the predictor</td>
</tr>
<tr>
<td>OM</td>
<td>Overall Mispredictions - number of overall mispredictions</td>
<td>the prediction (either local or global) is incorrect</td>
</tr>
<tr>
<td>LM</td>
<td>Local Mispredictions - number of local mispredictions</td>
<td>the local predictor is selected and makes an incorrect prediction</td>
</tr>
<tr>
<td>GM</td>
<td>Global Mispredictions - number of global mispredictions</td>
<td>the global predictor is selected and makes an incorrect prediction</td>
</tr>
</tbody>
</table>

Using these measurements, we can calculate the following metrics:

\[
\text{% local predictor used} = \frac{LF}{CF}
\]
\[
\text{% global predictor used} = \frac{GF}{CF}
\]

\[
\text{local misprediction rate} = \frac{LM}{LF}
\]
\[
\text{global misprediction rate} = \frac{GM}{GF}
\]
\[
\text{overall misprediction rate} = \frac{OM}{CF}
\]

Results

(shown on attached sheet in Tables 1-5)

Analysis

Frequency that the local predictor is used

According to Table 2, the percentage of times the local predictor is selected is higher in floating-point benchmarks since these programs contain a high number of loops which have a high probability of repetitive behaviors (that favor local predictions). For most of the integer benchmarks, the majority of the predictions do come from the local predictor, although not nearly at the same magnitude. This contrast can be attributed to the fact that the integer benchmarks contain less loops and more conditional branches which are harder to predict at runtime.
**Misprediction rates**
The overall misprediction rate (shown in Table 3) for the floating-point benchmarks is small, ranging from 1%-3%. Again, this is due to the fact that many of the branches in these benchmarks are loops containing repetitive computations. Because of this, the local predictor is used more often and is extremely accurate with an average misprediction rate of 0%-2% as shown in Table 4. The global misprediction rate (shown in Table 5) is slightly higher for the floating-point benchmarks, ranging from 2%-3%, but does not influence the overall rate as much since the global predictor is rarely used.

The overall misprediction rate for the integer benchmarks is higher, ranging from 7% for the mcf program to 13% for the crafty program. In all of the integer cases, the behaviors of the branches become much more unique to each program, and thus much more unpredictable. The local and global misprediction rates are also higher for the integer benchmarks, with ranges of 8% - 15% and 8% - 18% respectively. In particular, the misprediction rate for crafty is the highest in all 3 cases, probably due to the fact that the branch targets in this program are greatly influenced by the input to the program.

**Conclusion/Recommendations**
Based upon these findings, I would recommend that changes be made to the global predictor. The local predictor seems to be performing well with the floating-point benchmarks, so I would be hesitant to make any drastic changes to this portion of the hardware. Although the local predictor is less accurate among the integer benchmarks, it is also chosen less often which lessens its role in the overall misprediction rate.

I would suggest that several simulations be performed first before investing in any expensive hardware modifications. The changes that should be simulated during future testing are:

* Increasing the number of entries in the global predictor table - Adding additional history bits to the entries in this table may make global predictions more accurate. It would also be interesting to note the limitations of this option (i.e. at what point any further increases would not be helpful). I would suggest running further tests with a global predictor table of 6K and 8K entries first to see if any improvements are gained with this approach.

* Changing the scheme behind the 2-bit counter in each entry of the global predictor table - Currently, a standard 2-bit predictor is used in this table. It may be useful to determine if there are other schemes (based upon different finite state automata) which could lead to greater accuracy. I would suggest making slight modifications to the saturating counter scheme to see if this could lead to improved predictions.

When conducting simulations, different models with the changes described above should be used, as well as combinations of the listed changes. The same group of benchmarks should be used during simulation in order to make it easier to compare the results listed from these findings to the results taken from simulation experiments. I would be more inclined to make changes of the second type listed above since it will probably be easier (and less expensive) to change the transition logic of the counter instead of adding to the hardware by increasing the size of the global predictor table. I would only modify the size of the global predictor table if significant improvements (i.e. lower misprediction rates by 3% or more) were gained during simulation.