2000 p8q4
IAP

Comparative Architectures / IAP Q2 Extended Answer

Outline the design of a simple dynamic branch prediction cache which would provide a high degree of accuracy for branches that exhibit a strong bias in one or other direction (for example, loop-closing branches). [marks 5]

Assume a RISC-style processor with 32 bit instructions and a 512 entry cache.

The cache will be indexed by bits 2-10 of the branch instruction's address. We could hash other address bits to reduce the possibility of aliasing, but using the significant low order bits of the PC will probably work OK. The cache does not need tags as it does not affect program correctness, only performance.

A two bit bi-modal predictor will be used. Each of the 512 entries will consist of two bits of SRAM, which will store the value of a 2 bit saturating counter.

In parallel with the instruction decode stage, the prediction cache is accessed using the PC of the current instruction. If the instruction turns out not to be a branch the output of the predictor is ignored and no update is made to the predictor's state. If the instruction is a branch, the branch is predicted as taken if the counter value in the cache entry associated with this PC is equal to 2 or 3. Instruction fetch proceeds according to the prediction.

Once the true out come of the branch is known (much later in the pipeline), the state of the prediction cache is updated. A saturating counter is used to update the branch's cache entry, by incrementing it if the branch was taken, decrementing it if it was not. The counter saturates at 3 and 0 respectively.

Using a two bit counter provides greater accuracy than one bit as the predictor does not get confused when a branch makes a single deviation from its normal behaviour. e.g. loop closing branches will typically only get mispredicted upon loop exit. A single bit counter is likely to also get the next execution of the branch wrong.

Give an example of a simple code sequence containing branches that would be mispredicted by your design, and describe how it would behave. [marks 5]

Branches that are alternately taken, not-taken will be mispredicted by this scheme.

```
for(i=0;;i++)
  {
    if( i&1 = 0 )  // frequent mispredict
      {
        ...
      }
  }
```

Assuming no aliasing, the counter value will continually oscillate. Depending on its starting value, it will oscillate between 0 and 1, 1 and 2, or 2 and 3. The first and last of these possibilities yield a 50 percent mispredict rate, whereas the middle case yields a dismal 100 percent mispredict rate.

Revise your prediction cache design to make use of a four bit 'local history' for each entry. [marks 5]

Each cache entry will now consist of a 4 bit pattern containing the branch's recent history, and sixteen two bit saturating counter values.

The history field is effectively implemented as a shift register. It is updated when the branch outcome is known, a 1 being shifted in if the branch is taken, a 0 otherwise.

When making a prediction, the shift register value is used to select which of the 16 two bit saturating counters to access (states 2 and 3 predict a taken branch). When the branch outcome is know the same counter is updated as per the bimodal predictor scheme. The branch history field is also updated.

How would this predictor perform with a branch exhibiting the following repeating sequences? (T = taken, N = not-taken)

```
NTTNNTTNNTTN...
```

The first is effectively a repeating sequence of length 4. It will be predicted perfectly, using only four of the 16 counters. Assuming new history values are shifted into the right of the register:

0110 $\Rightarrow$ 0 not taken
1100 $\Rightarrow$ 1 taken
1001 $\Rightarrow$ 1 taken
0011 $\Rightarrow$ 0 not taken

```
NTTNNTNTTNNT...
```

This is a sequence of length 6, but is still correctly predicted as all sub-sequences of length 4 are unique:

0110 $\Rightarrow$ 0
1100 $\Rightarrow$ 1
1001 $\Rightarrow$ 0
0010 $\Rightarrow$ 1
0101 $\Rightarrow$ 1
1011 $\Rightarrow$ 0

```
TNTTTTTNTTTT...
```

This is a sequence of length 6, but will be mispredicted as the 1111 subsequence appears twice:

1011 $\Rightarrow$ 1
0111 $\Rightarrow$ 1
1111 $\Rightarrow$ 1 **
1111 $\Rightarrow$ 0 **
1110 $\Rightarrow$ 1
1101 $\Rightarrow$ 1

The 1111 counter value will oscillate. Depending on initial conditions, the mispredict rate for this branch will be either 1/3 or 1/6.