Comparative Architectures 2004 – Paper 8 Question 1 (IAP)

(a) Modern processors typically support out-of-order execution windows that can extend for several 10s of instructions. Branches are commonplace, and hence the speculation must extend across multiple branches, with compounded probability of mis-prediction. In the event of a mispredict being detected, computation must be backtracked to the branch, potentially resulting in significant amounts of work being discarded, in addition to the time to begin execution from the correct destination.

(b) Local history branch predictors work well for predicting the behaviour of branches that follow repeated (relatively short) sequences. In principle, they examine the successive behaviour of the branch in question in isolation (though aliasing in the prediction cache can lead to complications).

Many different implementation are possible, with different silicon area/accuracy trade offs. The Pentium III uses the significant low-order address bits of the branch's PC value to index the prediction cache and hence record the state for that branch. One field acts as a shift register that records the branch's previous four outcomes (updated when the outcome is known). This four bit value is used to select between sixteen two-bit saturating counters that are used as per a simple bi-modal predictor to provide a prediction (e.g. 0,1 taken; 2,3 not taken; once outcome is known, decrement if taken, increment if not taken).

Such a predictor can accurately track any pattern in which all subsequences of length four are unique. This is true for all patterns of length 6 or less, and some patterns of up to length 16.

Since many branches have just simple sequences (all taken, not taken, or perhaps alternate), many of the two-bit counters will not be being used, and hence the Si area may not be being used effectively. Alpha EV6 addresses this by recording a 10 bit history for each branch, then using just the history to index a single array of 1024 three-bit counters shared between all branches. Hence, branches following the same pattern will share the same counters in the array. Problems could occur if there are branches with complex patterns that alias in subsequences of length 10, but these will be rare (use of a three bit counter goes some way to prevent these events from perturbing the prediction of common patterns).

(c) Some branches are hard to predict because they are not strongly biased and either follow very long sequences, or they are driven by data that causes their behaviour to be near random.

In such instances, branch elimination is a good strategy. Predicated execution can enable both branches of an if/else statement to be evaluated without side effects. Conditional move can be used to approximate the same behaviour, evaluating both branches then committing one. Care must be taken to ensure their are no side effects on the 'wrong' path.

(*d*) The Pentium IV employs a 12k uop trace cache to optimise instruction dispatch. The x86 instructions are pre-decoded and stored as uops in the trace cache. A single x86 instruction may map to multiple uops.

uops in the trace cache are arranged in order of the predicted path of execution, hence representing the instructions executed in a trace that extends across branches (rather than just being in program PC order).

Packing decoded instructions in this manner means that dispatch is very efficient as a block of instructions contains no 'holes' at the head or tail, as would occur with a conventional cache due to branches and branch targets.

(*e*) Runtime binary rewriting can be used to provide online optimisation of the code the CPU is executing. The rewriter could be part of an application runtime, part of the operating system, or operate below the operating system and hence be completely transparent (c.f. Transmeta). The optimiser could be driven by feedback from the CPU's performance counters. Regularly executed code sequences could be identified, and then traced through binary instrumentation or interpretation (some CPUs make this process easier by having performance counters that record the branch history).

Armed with a trace of program execution, the optimiser could create a new code segment (typically in a new area of memory rather than in-place) that linearises the instruction sequence. Entry points to the original sequence could then be patched such that the new sequence is called. Furthermore, this technique enables peep-hole optimisation of the linearised code sequence, possibly resulting in the elimination of the load part of some register spill/fills (the store may still be necessary for correctness).