LSU EE 4720 Homework 4 Solution Due: 25 April 2007

Problem 1: Estimate performance of the 8-way superscalar statically scheduled MIPS implementations described here. All are five stages, as used in class, and always hit the cache, as has been the case in class so far. Some of the implementations have no fetch group alignment restrictions, which means any eight contiguous instructions can be fetched. Some impose a fetch group alignment restriction, meaning if a CTI target is address a `IF` will fetch eight instructions starting at address $a\prime = 8 \times 4 \times \lfloor \frac{a}{8\times 4} \rfloor$ (for those preferring C: `aa = a & ~0x1f`). Instructions in $[a\prime, a)$ (or from `aa` to before `a`) will be squashed before reaching `ID`.

The implementations include a branch predictor that predicts when a branch is in `IF`, resolves (checks the prediction) when a branch is in `ID`, and if necessary recovers (squashes wrong-path instructions) when a branch is in `EX`. A branch is predicted when it is in `IF` and the prediction is used in the next cycle. Example 1, below, illustrates a correct taken prediction. The correctness of the prediction is checked, resolved, when the branch is in `ID`; if incorrect the wrong-path instructions are squashed and the correct path instructions are fetched in the next cycle (when the branch is in EX). This is illustrated in Example 2 for an incorrect taken prediction.

Note that due to alignment restrictions (if imposed) and branch placement the number of useful instructions fetched in a cycle can vary, and that is something to take into account in the subproblems below. The examples below illustrate *when* instructions will be fetched and squashed but they do not show *how many* will be fetched in every situation.

```
 # Example 1: Branch correctly predicted taken.  Target fetched in next cycle.
 #
 # Cycle           0  1  2  3  4  5  6
 beq  r1,r2, TARG   IF ID EX ME WB
 nop                IF ID EX ME WB
 ...

TARG:
 add r3, r4, r5        IF ID EX ME WB


 # Example 2: Branch wrongly predicted taken.
 # Target squashed, correct path (fall through) fetched in cycle after ID.

 # Cycle           0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
 beq  r1,r2, TARG   IF ID EX ME WB
 nop                IF ID EX ME WB
 sub r6, r7, r8           IF ID EX ME WB

TARG:
 add r3, r4, r5        IFx
 # Cycle           0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
```

All implementations run the same program, which has not been specially compiled for the 8-way machine. In the program, which has no floating-point instructions, any two data-dependent instructions have at least seven instructions between them. This avoids some stalls, assume that there are no other stalls in the superscalar implementation **due to data dependencies**.

Let n_i denote the number of dynamic instructions in the program and let n_b denote the number of dynamic instructions that are branches. For the questions below show answers in terms of these symbols and also show values for $n_\text{i} = 10^{10}$ and $n_\text{b} = 2 \times 10^9$. Assume that half of the times a branch is executed it is taken.

(*a*) Suppose the 8-way implementation has perfect branch prediction and has no fetch alignment restrictions. Approximately how long (in cycles) will it take to run the program. State any assumptions. *Hint: Because* *of the branches it's* $> \frac{n_i}{8}$.

First of all, if it weren't for branches execution would take $\frac{n_i}{8}$ cycles. Performance is lost when a branch is not in the penultimate slot in a group. For example, suppose a taken branch is the first of eight instructions in ID. Six of those eight would have to be squashed. Because there is perfect branch prediction none of the instructions in IF are squashed. Only when a taken branch is in the last slot do instructions in IF get squashed, the seven following the delay-slot instruction. If a taken branch is in position i then the number of squashed instructions is $\begin{cases} 6-i & i<7 \\ 7 & i=7 \end{cases}$, where $i=0$ is the first slot.

Since the program has not been specially compiled the branch is equally likely to be in any slot, so the average number of squashed instructions for a taken branch is 3.5.

The execution time is then $\boxed{\left(n_i + \frac{n_b}{2}3.5\right)/8 \text{ cycles}}$. For the sample numbers the execution time is $\boxed{1.69 \times 10^9 \text{ cycles}}$.

(*b*) Repeat the question above for a predictor that always predicts not taken (which essentially means no predictor).

If a branch outcome is not taken no instructions are squashed. If a branch in ID is taken then seven or eight instructions in IF must be squashed (seven when the branch is in the last slot). The number of squashed instructions for a branch in slot i is $8+6-i$ for $0 \le i < 8$. The average is 10.5 and so the execution time is $\boxed{\left(n_i + \frac{n_b}{2}10.5\right)/8 \text{ cycles}}$, for the sample numbers the execution time is $\boxed{2.56 \times 10^9 \text{ cycles}}$. Note the difference between perfect branch prediction (previous subproblem) and no branch prediction.

(*c*) Repeat the question above for a predictor with a 95% prediction accuracy. (Yes, that means 95% of the predictions are correct.)

There are four cases to consider. Predicted not taken, outcome not taken (call that nn), nt, tn, and tt. In case nn nothing is squashed. Case nt is what was analyzed in part b, the average number of squashes was 10.5. Case tt was analyzed in part a, the average number of squashes was 3.5.

For this part we need to find the number of squashes when a branch is predicted taken but its not (the tn case). Consider such a branch when it is in ID and assume the branch is not in slot 7. Because the branch was predicted taken $6-i$ instructions in ID will be squashed. When the branch is resolved not taken all the instructions in IF (which are on the taken path) will be squashed. The total number of squashed instructions is then $8+6-i$ for $i<7$. It is possible to actually not squash the instructions in ID in this case, but here we will assume its too difficult. (For one, because the branch is resolved at the end of ID so they'd have to be squashed in EX in the tt case.)

If the branch is in slot 7 then IF will be fetching the delay slot instruction along with 7 which based on the prediction are to be squashed. If the hardware is smart enough these can be saved and so that zero instructions are squashed in the tn case if the branch is in slot 7. The average number across all slots is 9.625 instructions squashed.

Assuming prediction accuracy is the same for taken- and not-taken branches one can find a weighted average over the four cases:

$$\overbrace{0 \times \frac{0.95}{2}}^{\text{nn case}} + \overbrace{10.5 \times \frac{0.05}{2}}^{\text{nt case}} + \overbrace{9.625 \times \frac{0.05}{2}}^{\text{tn case}} + \overbrace{3.5 \times \frac{0.95}{2}}^{\text{tt case}} = 2.17 \text{ cycles.}$$

The execution time is $\boxed{\left(n_i + n_b 2.17\right)/8 \text{ cycles}}$, for the sample numbers the execution time is $\boxed{1.52 \times 10^9 \text{ cycles}}$. Note that this calculation uses the total number of branches, not just the taken ones.

(*d*) Once again, suppose the 8-way implementation has perfect branch prediction but now fetch is restricted to aligned groups. Approximately how long (in cycles) will it take to run the program. State any assumptions.

Each time a branch is taken 3.5 instructions near the branch will be squashed (see part a). Because fetch isn't aligned, on average 3.5 instructions at the target will be squashed, for a total of 7 instructions.

The execution time is $\boxed{\left(n_i + \frac{n_b}{2}7\right)/8 \text{ cycles}}$, for the sample numbers the execution time is $\boxed{2.13 \times 10^9 \text{ cycles}}$.

Problem 2: Consider a bimodal branch predictor with a 2^{10}-entry branch history table (BHT).

(a) What is the prediction accuracy on the branch below with the indicated behavior assuming no interference. Assume that the pattern continues to repeat. Provide the accuracy after warmup.

```
0x1000 beq r1, r2  TARG    t  t  t  n  t  t  n  n  n  t  t  t  n  t  t  n  n  n  ...
```

The counter values, prediction and outcome are shown below. Note that the counter values will repeat. The prediction accuracy is $4/9 = .444$, which is not good at all.

```
        Counter:        0  1  2  3  2  3  3  2  1  0
0x1000 beq r1, r2  TARG   t  t  t  n  t  t  n  n  n  t  t  t  n  t  t  n  n  n  ...
      Prediction         n  n  t  t  t  t  t  t  n
      Prediction wrong:   X  X     X        X  X
```

Problem 3: Suppose that for some crazy reason it's important that the branch at address `0x1000` be predicted accurately, even if that means suffering additional mispredictions elsewhere. The result of this crazyness is the code below, in which the branch in `HELPER` is intended to help the branch at `0x1000`.

(a) Choose an address for `HELPER` so that `0x1000` is helped.

Choose the address so that `HELPER` and the branch at `0x1000` use the same entry in the BHT. (This isn't something that's ordinarily done.) Since the BHT has 2^{10} entries the address of `HELPER` must match `0x1000` in the lower 12 bits. One possible address is `0x2000`.

(b) Given a correct choice for the address of `HELPER`, find the prediction accuracy of the branch at `0x1000`.

```
  jal HELPER
  nop
0x1000:
  beq r1, r2  TARG   t  t  t  n  t  t  n  n  n  t  t  t  n  t  t  n  n  n  ...
  ...
  ....

HELPER:
  beq r1, r2 SKIP
  nop
SKIP:
  jr r31
  nop
```

The solution is worked out below. The counter is modified twice, before and after the branch (both times using its outcome). The accuracy is now $6/9 = .667$, better but still not that good.

```
        Counter:       01  23 33 32 12 33 32 10 00 01
0x1000 beq r1, r2  TARG   t  t  t  n  t  t  n  n  n  t  t  t  n  t  t  n  n  n  ...
      Prediction         n  t  t  t  t  t  t  n  n
      Prediction wrong:   X        X        X
```