CSCI 524 – Computer Architecture
Homework 5

Due: November 1, 2016

Alexander Powell

1. (a) **Assuming there is no forwarding hardware support, indicate the data hazards and insert nop instructions to eliminate them.**

   Because the lw instructionis placing a value in the R4 register, the sw instruction needs to be stalled twice before it can decode the instruction. There are no dependencies associated with the add instruction so no NOPs are needed after the sw instruction. Without forwarding, the instructions should look like this:

   ```
   lw R4, -100(R16)
   NOP
   NOP
   sw R2, 8(R4)
   add R2, R4, R4
   ```

   The pipelined stages will look something like:

   ```
   - - - - - - - - - - - - - -
   | F D X M W               |
   |   - - - - -             |
   |     - - - - -           |
   |       F D X M W         |
   |         F D X M W       |
   - - - - - - - - - - - - - -
   ```

   (b) **Assuming there is full forwarding hardware support, indicate the data hazards and insert nop instructions to eliminate them.**

   With data forwarding, after the memory writeback stage in the lw instruction, that result can be immediately forwarded to the memory write stage in the sw instruction as well as the execute stage in the add instruction. This means that there will only be one NOP instruction between the load and the store. Data will be forwarded immediately after it is available from the execute phase in the lw.

   ```
   lw R4, -100(R16)
   NOP
   sw R2, 8(R4)
   add R2, R4, R4
   ```

2. (a) **What is the accuracy of always-taken and always-not-taken predictors for this sequence of branch outcomes?**

   When the predictor assumes the branch will always be taken, the accuracy for this pattern is 20%. Conversely, using an always-not-taken predictor gives an accuracy of 80%.

   (b) **What is the accuracy of a one-bit predictor assuming the predictor starts off in the predict-not-taken state? What is the accuracy of this one-bit predictor if this pattern is repeated forever?**

By using a one-bit predictor we get an accuracy of 60% for the pattern. The table below shows the thought process of the predictor:

Action	NT	NT	T	NT	NT
Predict	✓	✓	×	×	✓

If this pattern is repeated forever the accuracy will still be 60%.

(c) **What is the accuracy of a two-bit predictor for the first four branches in this pattern, assuming that the predictor starts off in the strongly predict-not-taken state? What is the accuracy of this two-bit predictor if this pattern is repeated forever?**

Using a two-bit predictor on the first four branches of the sequence will give us an accuracy of 75%. The table below shows how the predictor would handle the branches. Note that the predictor always predicts NT because the two-bit predictor would require two Ts to change the prediction to taken.

Action	NT	NT	T	NT
Predict	✓	✓	×	✓

Again, if the pattern were repeated forever the accuracy would still be 75%.

3. The following tables show the performance evaluations for the six different benchmarks: bzip2, mcf, hmmer, sjeng, milc and equake.

Performance Results for bzip2

	M1	M2	M3
# Instructions	2000000	2000000	2000000
CPI	2.6732	4.0431	4.7278
Il1 miss rate	0.0163%	0.0163%	0.0163%
Dl1 miss rate	52%	52%	52%
UL2 miss rate	31%	31%	31%

Performance Results for mcf

	M1	M2	M3
# Instructions	2000000	2000000	2000000
CPI	2.3861	2.4903	2.7963
Il1 miss rate	2.5%	2.5%	2.5%
Dl1 miss rate	11%	11%	11%
UL2 miss rate	19.6%	19.6%	19.6%

Performance Results for hmmer

	M1	M2	M3
# Instructions	2000000	2000000	2000000
CPI	2.3601	2.4550	2.7139
Il1 miss rate	0.28%	0.28%	0.28%
Dl1 miss rate	8.4%	8.4%	8.4%
UL2 miss rate	13.7%	13.8%	13.8%

Performance Results for sjeng

	M1	M2	M3
# Instructions	2000000	2000000	2000000
CPI	2.2704	2.5091	2.7270
Il1 miss rate	0.006%	0.006%	0.006%
Dl1 miss rate	5.1%	5.1%	5.1%
UL2 miss rate	21%	21%	21%

Performance Results for milc

	M1	M2	M3
# Instructions	2000000	2000000	2000000
CPI	2.3309	2.4885	2.5363
Il1 miss rate	0.38%	0.38%	0.38%
Dl1 miss rate	12.2%	12.2%	12.2%
UL2 miss rate	24.7%	24.7%	24.7%

Performance Results for equake

	M1	M2	M3
# Instructions	2000000	2000000	2000000
CPI	2.6510	2.8505	3.5300
Il1 miss rate	5%	5%	5%
Dl1 miss rate	21.5%	21.5%	21.5%
UL2 miss rate	2.4%	2.4%	2.4%

If the clock rate for the first machine is 250 MHz (or 0.25 GHz), the second machine is 1.5 GHz, and the third is 2 GHz, the execution times for the bzip2 bechmark can be computed as

M1: $\dfrac{2000000 \times 2.6732}{0.25} = 21385600$ nsec.

M2: $\dfrac{2000000 \times 4.0431}{1.5} = 5390800$ nsec.

M3: $\dfrac{2000000 \times 4.7278}{2} = 4727800$ nsec.

The table below shows the execution times and geographic means (in nanoseconds) of the integer benchmarks.

Benchmark	**M1 Time**	**M2 Time**	**M3 Time**
bzip2	21385600	5390800	4727800
mcf	19088800	3320400	2796300
hmmer	18880800	3273333	2713900
sjeng	18163200	3345466	2727000
Geometric Mean	1.934×10^7	3.742×10^6	3.154×10^6

The table below shows the execution times and geographic means (in nanoseconds) of the floating point benchmarks.

Benchmark	**M1 Time**	**M2 Time**	**M3 Time**
milc	18647200	3318000	2536300
equake	21208000	3800666	3530000
Geometric Mean	1.989×10^7	3.551×10^6	2.992×10^6

So, it's clear that the third machine has a better performance for both the integer and floating point benchmarks. Even though this machine has higher cache latencies, because its clock rate is higher its exection time is improved.

4. The following tables show information about each benchmark and each machine, including number of branches executed, total number of instructions committed, and the execution time (in nanoseconds).

Performance Results for bzip2

	M1	M2	M3
# Branches Executed	237170	237224	212047
# Instructions Committed	2000000	2000000	2000000
# Branches Committed	212047	212047	212047
CPI	1.9681	1.9679	1.8918
Performance (nsec)	0.30×10^7	0.30×10^7	0.28×10^7

Performance Results for mcf

	M1	M2	M3
# Branches Executed	444242	450023	271779
# Instructions Committed	2000000	2000000	2000000
# Branches Committed	271779	271779	271779
CPI	1.6235	1.6199	1.0653
Performance (nsec)	0.24×10^7	0.24×10^7	0.16×10^7

Performance Results for hmmer

	M1	M2	M3
# Branches Executed	533188	534517	296514
# Instructions Committed	2000000	2000000	2000000
# Branches Committed	296656	296645	296514
CPI	1.4288	1.4272	0.7961
Performance (nsec)	0.21×10^7	0.22×10^7	0.12×10^7

Performance Results for sjeng

	M1	M2	M3
# Branches Executed	455282	469122	322279
# Instructions Committed	2000000	2000000	2000000
# Branches Committed	322279	322279	322279
CPI	1.5461	1.5380	0.8875
Performance (nsec)	0.23×10^7	0.23×10^7	0.13×10^7

Performance Results for milc

	M1	M2	M3
# Branches Executed	351247	351084	190530
# Instructions Committed	2000000	2000000	2000000
# Branches Committed	190530	190530	190530
CPI	1.5906	1.5895	1.1453
Performance (nsec)	0.24×10^7	0.24×10^7	0.17×10^7

Performance Results for equake

	M1	M2	M3
# Branches Executed	555871	571885	378285
# Instructions Committed	2000000	2000000	2000000
# Branches Committed	378285	378285	378285
CPI	1.8267	1.8174	1.2366
Performance (nsec)	0.27×10^7	0.27×10^7	0.19×10^7

So, once again, it appears that the third machine has the best performance results. The first two machines don't seem to differ very much between their execution times. It makes sense that M3 is the fastest because it was the machine using the perfect branch prediction strategy. Therefore, we can conclude that good branch predictions are an important part of an architecture, since M3 even had larger latencies that M1 and M2 but still out-performed them due to the perfect branch prediction strategy.