CS M151B - Homework 7

Kuruvilla Cherian, Jahan
UID: 104436427
Section 1B

March 11, 2017

5.3

1

To calculate the cache block size, we need to consider the offset.

$$cache\_block\_size = 2^{offset} = 2^5 = \textbf{32 bytes}$$

$$\implies cache\_block\_size = \frac{32bytes}{4bytes} = \textbf{8 words}$$

2

The total number of entries correspond to the to number of indices available.

$$number\_entries = 2^{indices} = 2^5 = \textbf{32 entries}$$

4

Address	Index	Hit/Miss	Replaced
0	00000	Miss	N
4	00000	Hit	N
16	00000	Hit	N
132	00100	Miss	N
232	00111	Miss	N
160	00101	Miss	N
1024	00000	Miss	Y
30	00000	Miss	Y
140	00100	Hit	N
3100	00000	Miss	Y
180	00101	Hit	N
2180	00100	Miss	Y

Table 1: Table representing the replacement and hit/miss ratios

We see that overall there are **4 replacements**

5

Looking at Table 1, we can see the overall number of misses were 8, and 4 hits resulting in a **1:2 hit/miss ratio** with a hit percentage of **33%**.

5.5

1

We have the following sequence:

0, 2, 4, 6, 8, 10, 12, 14, 16, ...

This shows that our access is per 2 bytes of information. Our direct-mapped cache is **64 KiB** in size with **32 byte** blocks, which means there are in total $\frac{64 \cdot 1024}{32} = \mathbf{2048}$ blocks. This implies $log_2 2048 = 11$ index bits, with $log_2 32 = 5$ offset bits. Because the block is itself 32 bytes, and we pull in 2 bytes per sequence index, we notice that in one grab a block can hold $\frac{32}{2} = \mathbf{16}$ addresses per fetch. The first address of each fetch from the sequence will be a miss since it will be the first time we are seeing this address, given the nature of continual sequence. Thus our miss rate is:

$$\frac{1}{16} \cdot 100 = 6.25\%$$

The categorization of these misses are **Compulsory Misses** because there is no way of getting around the problem of never having seen the values before. This rate is sensitive to the sequence jumps and the overall size of each block.

2

The general equation for miss rate in this instance would be:

$$miss\_rate = \frac{jump\_difference}{block\_size}$$

Thus we have the following changes:

$$miss\_rate_1 = \frac{2}{16} = \frac{1}{8} \implies \mathbf{12.5\%}$$

$$miss_rate_2 = \frac{2}{64} = \frac{1}{4} \implies \mathbf{3.125\%}$$

$$miss_rate3 = \frac{2}{128} = \frac{1}{64} \implies \mathbf{1.5625\%}$$

This method due to its consistent sequence grabs acts similar to accessing large chunks from an array and thus exhibits more of **spatial** locality.

5.6

0.1 2

	L1 Size	L1 Miss Rate	L1 Hit Time
P1	2KiB	8.0%	0.66ns
P2	4KiB	6.0%	0.90ns

Table 2: Table representing the first level memory information

The general equation for the AMAT is:

$$AMAT = hit\_time + (miss\_rate \cdot miss\_penalty)$$

In this case the miss time is effectively 0. We must also take into account the fact that memory accesses are **36%** of all instructions and take **70ns** to access. Given this information we can calculate the AMAT for each processor using the aforementioned equation as follows:

$$AMAT_{P1} = 0.66 + (0.08 \cdot 70) = \mathbf{6.26ns}$$

$$AMAT_{P2} = 0.90 + (0.06 \cdot 70) = \mathbf{5.1ns}$$

4

L2 Size	L2 Miss Rate	L2 Hit Time
1MB	95%	5.62ns

Table 3: Table representing the second level memory information

Using the same information as before and the new level in memory hierarchy, we have the following:

$$AMAT_{P1_{new}} = 0.66 + (0.06 \cdot ((5.62 + (0.95 \cdot 70))) = \textbf{6.43ns}$$

Since **6.43ns** > **6.26ns**, we see that the added secondary cache makes memory access times worse on average.

Exam Question - TCPI

a

In this case we are only concerned about the AMAT for the data cache memory accesses upon misses. The general equation for AMAT is:

$$AMAT = hit\_time + (miss\_rate \cdot miss\_penalty)$$

For the data cache we see that the *hit_ rate* is **1** cycle. As specified by the problem, upon a miss on the data cache, we only suffer the stall of 1 cycle, thus implying that the *miss_ penalty* is **1** cycle at the level of the data cache. The rate of miss is a cascading series depending on the number of cycles and miss rate at each level of the memory hierarchy. We use the values from the question to get the following answer:

$$AMAT = 1.0 + 0.3 \cdot (10 + 0.2 \cdot 80) = \textbf{8.8 cycles}$$

b

$$TCPI = BCPI + MCPI$$

$$BCPI = CPI_{Peak} + CPI_{Data\_Hazard} + CPI_{Control\_Hazard}$$

$$CPI_{Hazard} = Frequency \cdot Penalty$$

$$\therefore BCPI = 1.0 + (0.2 \cdot 0.6 \cdot 1) + (0.3 \cdot 0.5 \cdot 1) = 1.27$$

$$MCPI = Stall_{Instruction} + Stall_{Data}$$

$$\therefore MCPI = (1.0 \cdot (0.1) \cdot (10 + 0.2 \cdot 80)) + (0.2 \cdot (0.3) \cdot (10 + 0.2 \cdot 80)) = 4.16$$

$$\therefore TCPI = 1.27 + 4.16 = \textbf{5.43}$$

c

Replacing the CPI with the TCPI from before we can see that the old ET was:

$$ET = TCPI_{old} \cdot 10^6 = 5.43 \cdot 10^6 cycles$$

. Given that we have $1,000,000$ instructions and have the given percentage of loads, we can look at the old versus new instruction counts as shown in Table 4:

	Old IC	New IC	New %
branch	300,000	200,000	22.2
load/store	200,000	200,000	22.2
r-type	500,000	500,000	55.5
total	1,00,000	900,000	100

Table 4: Table representing the instruction count

Even though we have $\frac{1}{6}$ of branches to be procedure calls, we notice that each *jal* results in a one-to-one relationship to *jr* instructions, meaning that $\frac{2}{6}$ of branches are removed by inlining, which is why Table 4, shows us that $IC_{branch_{new}} = IC_{branch_{old}} \cdot \frac{6-2}{6} = 200,000$ instructions.

We noticed that 50% of branches were being taken (i.e. $150,000$ instructions), but now that we don't have anymore jal or jr instructions. Since these instructions represented $\frac{1}{6}$th of all the branch instructions, we have that $\frac{1}{6} \cdot 300000 = 50000$ branches always being taken, thus meaning that given our predictor is a static not taken predictor we have a branch miss rate of :

$$miss\_rate_{branch} = \frac{50000}{200000} = 25\%$$

Now given our equations from earlier sections we calculate the new BCPI and MCPI to calculate the new TCPI to then calculate the Execution time which can lead to finding the miss rate increase for instructions.

$$MCPI_{new} = (1.0 \cdot (miss\_rate_{I-cache}) \cdot (10 + 0.2 \cdot 80)) + (0.2 \cdot (0.3) \cdot (10 + 0.2 \cdot 80))$$

$$\implies MCPI_{new} = (miss\_rate_{I-cache}) \cdot 26 + 1.73$$

$$BCPI_{new} = 1.0 + (0.2 \cdot 0.6 \cdot 1) + (0.2 \cdot 0.25 \cdot 1) = 1.18$$

$$TCPI_{new} = BCPI_{new} + MCPI_{new}$$

$$\therefore TCPI = 1.18 + (miss\_rate_{I-cache}) \cdot 26 + 1.73$$

$$ET_{new} = TCPI_{new} \cdot (9 \cdot 10^5) \implies (2.92 + (miss\_rate_{I-cache}) \cdot 26) \cdot (9 \cdot 10^5)$$

$$\implies 2.63 \cdot 10^6 + (miss\_rate_{I-cache}) \cdot 2.34 \cdot 10^7$$

The key insight is that if we wanted to see our maximal instruction cache miss rate then we set the new execution time to the old one as follows:

$$ET_{new} = ET_{old}$$

$$\implies 2.63 \cdot 10^6 + (miss\_rate_{I-cache}) \cdot 2.34 \cdot 10^7 = 5.43 \cdot 10^6$$

$$\therefore miss\_rate_{I-cache} = \mathbf{0.1197}$$

This means that our Instruction cache miss rate can now be increased to $11.97 \approx \mathbf{12\%}$.