CS152 Spring 2008 Quiz 2 Answer Key

Problem Q2.1: Victim Cache Evaluation

Problem Q2.1.A **Baseline Cache Design**

Component	Delay equation (ps)		FA (ps)
Comparator	200×(# of tag bits) + 1000		6800
N-to-1 MUX	$500 \times \log_2 N + 1000$		1500
Buffer driver	2000		2000
AND gate	1000		1000
OR gate	500		500
Data output driver	500×(associativity) + 1000		3000
Valid output driver	1000		1000

Table Q2.1-1

The **Input Address** has 32 bits. The bottom two bits are discarded (cache is word-addressable) and bit 2 is used to select a word in the cache line. Thus the **Tag** has 29 bits. The **Tag+Status** line in the cache is 31 bits.

The **MUX**es are 2-to-1, thus N is 2. The associativity of the **Data Output Driver** is 4 – there are four drivers driving each line on the common **Data Bus**.

Delay to the **Valid Bit** is equal to the delay through the **Comperator**, **AND** gate, **OR** gate, and **Valid Output Driver**. Thus it is 6800 + 1000 + 500 + 1000 = 9300 ps.

Delay to the **Data Bus** is delay through MAX ((**Comperator**, **AND** gate, **Buffer Driver**), (**MUX**)), **Data Output Drivers**. Thus it is MAX (6800 + 1000 + 2000, 1500) + 3000 = MAX (9800, 1500) + 3000 = 9800 + 3000 = 12800 ps.

Critical Path Cache Delay: 12800 ps

Problem Q2.1.B **Victim Cache Behavior**

Input Address	Main Cache									Victim Cache		
	L0	L1	L2	L3	L4	L5	L6	L7	Hit?	Way0	Way1	Hit?
	inv	inv	inv	inv	inv	inv	inv	inv	-	inv	inv	-
00	0								N			N
80	8								N	0		N
04	0								N	8		Y
A0			A						N			N
10		1							N			N
C0					C				N			N
18									Y			N
20			2						N		A	N
8C	8								N	0		Y
28									Y			N
AC			A						N		2	Y
38				3					N			N
C4									Y			N
3C									Y			N
48					4				N	C		N
0C	0								N		8	N
24			2						N	A		N

Table Q2.1-2

15% of accesses will take 50 cycles less to complete, so the average memory access improvement is 0.15 * 50 = 7.5 cycles.

Problem Q2.2: Code and Data Rearrangement

```
for(j=0; j < N; j++) {
     for(i=0; i < M; i++) {
        x[i][j] = 2 * x[i][j];
     }
 }

for(i=0; i < M; i++) {
     for(j=0; j < N; j++) {
        x[i][j] = 2 * x[i][j];
     }
 }
```

```
for(i=0; i < N; i++)
    a[i] = b[i] * c[i];
for(i=0; i < N; i++)
     d[i] = a[i] * c[i];

for(i=0; i < N; i++)
{
       a[i] = b[i] * c[i];
       d[i] = a[i] * c[i];
  }
```

Problem Q2.3: Caches and Memory Access Patterns

You have just accepted a position at Caches-R-Us as a research scientist. Your first task is to assess the pathological performance of various cache organizations. You decide to start by looking at two basic caches, both with a capacity of four words. The first is a direct-mapped cache with one word per cache line. The second is a fully-associative cache also with one word per cache line and an LRU replacement policy. For both of the following questions assume the caches are initially empty, i.e., all lines are invalid.

Problem Q2.3.A

Please specify a memory access pattern that will cause the fully-associative cache to incur fewer misses than the direct-mapped cache.

There are many simple memory access patterns that could cause conflict misses in the direct-mapped cache that would result in better performance from the fully-associative cache. An example would be an access pattern that had a stride of 4 and a range of 16. In this case, we would have an access pattern of 0, 4, 8, 16, 0, 4, 8, 16, etc. All the accesses would be mapped to the same cache line for the direct-mapped cache, resulting in a 100% miss rate. The fully-associative cache, on the other hand, would contain all four data words and have a 0% miss rate after a round of cold-start misses.

Problem Q2.3.B

Does there exist a memory access pattern that causes the direct-mapped cache to incur fewer misses than the fully-associative cache? If so, please give one such access pattern, or else explain why this is not possible.

Yes, there do exist memory access patterns that result in better performance from a direct-mapped cache. Although they are not common occurrences, we can construct a pathological memory access pattern that causes the fully-associative cache to kick out data that will be reused (because of the LRU replacement policy) but maps to different locations in the direct-mapped cache. An example is a simple loop that accesses 5 sequential memory words, such as 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, etc. In this case, the direct-mapped cache would miss on every access to words 0 and 4 but hit on words 1,2, and 3 after a round of cold start misses, resulting in a 40% miss rate. For the fully-associative cache, the LRU policy would cause the cache to kick out a cache line (since we are accessing more data words than the cache capacity) right before it will be used in the next access, resulting in a 100% miss rate.

Problem Q2.4: Cache Parameters Short Answer

Problem Q2.4.A

TRUE. Since cache size is unchanged, the line size doubles, the number of tag entries is halved.

Problem Q2.4.B

FALSE. The total number of lines across all sets is still the same, therefore the number of tags in the cache remain the same.

Problem Q2.4.C

TRUE. Doubling the capacity increases the number of lines from N to 2N. Address i and address i+N now map to different entries in the cache and hence, conflicts are reduced.

Problem Q2.4.D

FALSE. The number of lines doubles but the line size remains the same. So the compulsory "cold-start" misses stays the same.

Problem Q2.4.E

TRUE. Doubling the line size causes more data to be pulled into the cache on a miss. This exploits spatial locality as subsequent loads to different words in the same cache line will hit in the cache reducing compulsory misses.