CS61c Summer 2014 Discussion 7 – Caches

1 T:I:O Problems

An address breaks down int T:I:O.

- **Offset** - "column index" - location of address within block
- **Index** - "row index" - location (row) of block within cache
- **Tag** - "block identifier" - is this the right block?

Fill out the following table. Assume all caches are write-back.

Address Bits	**Cache Data Size**	**Block Size**	**Tag Bits**	**Index Bits**	**Offset Bits**	**Total Row Bits**
16	4 KiB	4 B	4	10	2	38
32	32 KiB	16 B	17	11	4	147
32	64 KiB	16 B	16	12	4	146
64	2048 KiB	128 B	43	14	7	1069

Fill out the following table. Assume all caches are write-through.

Address Bits	**Cache Data Size**	**Cache Type**	**Block Size**	**Tag Bits**	**Index Bits**	**Offset Bits**	**Total Row Bits**
16	16 KiB	Direct Mapped	8 B	2	11	3	67
16	16 KiB	2-Way Set Associative	8 B	3	10	3	68
16	16 KiB	4-Way Set Associative	8 B	4	9	3	69
16	16 KiB	Fully Associative	8 B	13	0	3	78
32	64 KiB	Direct Mapped	16 B	16	12	4	145
32	64 KiB	Fully Associative	16 B	28	0	4	157
8	32 B	4-Way Set Associative	4 B	4	1	2	38

2 Cache hits and misses

Define the following cache terms:

1. **Cache hit** - requested address already in the cache (fast return)
2. **Cache miss** - cache index "empty" (not valid), so read from memory and write to cache (slow)
3. **Cache miss, block replacement** - cache index is valid but has wrong block (non-matching tag), so read from memory and override block (slow)

We have a byte-addressed computer, using a 32B cache with 8B blocks. The following byte memory addresses are accessed in order. Classify each access as a cache hit (**H**), miss (**M**), or miss with replacement (**R**).

1. `0x00000004` M
2. `0x00000005` H
3. `0x00000068` M
4. `0x000000C8` R
5. `0x000000DD` M
6. `0x00000045` R
7. `0x00000004` R
8. `0x000000C8` H

3 Analyzing C Code

```
#define NUM_INTS 8192
int A[NUM_INTS]; /* AT ADDRESS 0x100000 */
int i, total = 0;
for(i=0;i<NUM_INTS;i+=128) { A[i] = i; }      /* LINE 1 */
for(i=0;i<NUM_INTS;i+=128) { total += A[i]; } /* LINE 2 */
```

Let's say you have a byte-addressed computer with a total address space of 1MiB. It features a 16KiB CPU cache with 1KiB blocks.

1. How many bits make up a memory address on this computer? 20
2. What is the T:I:O breakdown? 6:4:10
3. Calculate the cache hit rate for the line marked Line 1: 50%. The integers are $4128 = 512$ bytes apart, which means that there are two accesses per block. The first access is a cache miss, but the second access is a cache hit, because A[i] and A[i + 128] are in the same cache block.
4. Calculate the cache hit rate for the line marked Line 2: 50%. At the end of line 1, we now have the second half of A inside our cache, so we get the same hit rate as before. Note that we do not have to consider cache hits for total, since the compiler will probably leave it in a register.
5. How could you optimize this computation? We could have fewer cache misses if we break up the loops to cover half the array (16 KiB) at a time. While this means that there would be 4 for loops in our C code, the hit rates of each loop would be 50%, 100%, 50%, 100% for an average hit rate of 75%.

4 Average Memory Access Time

AMAT is the average (expected) time it takes for memory access. It can be calculated using the following formula: **AMAT** = **hit time** + **miss rate** ∗ **miss penalty**. Remember that the miss penalty is the additional time it takes for memory access in the event of a cache miss. Therefore, a cache miss takes **hit time** + **miss penalty time**.

Suppose that you have a cache system with the following properties. What is the AMAT?

- L1$ hits in 1 cycle (local miss rate 25%)
- L2$ hits in 10 cycles (local miss rate 40%)
- L3$ hits in 50 cycles (global miss rate 6%)
- Main memory hits in 100 cycles (always hits)

The AMAT is $1 + 25\% * (10 + 40\% * (50)) + 6\% * (100) = 14.5$ cycles.