**HW2 Zhijia Chen**

# 1 B.2

Table 1: fully-associative cache

| Cache block | Set | Way | Possible memory blocks |
|---|---|---|---|
| 0 | 0 | 0 | M0, M1, ..., M31 |
| 1 | 0 | 1 | M0, M1, ..., M31 |
| 2 | 0 | 2 | M0, M1, ..., M31 |
| 3 | 0 | 3 | M0, M1, ..., M31 |
| 4 | 0 | 4 | M0, M1, ..., M31 |
| 5 | 0 | 5 | M0, M1, ..., M31 |
| 6 | 0 | 6 | M0, M1, ..., M31 |
| 7 | 0 | 7 | M0, M1, ..., M31 |

Table 2: four-way set associative cache

| Cache block | Set | Way | Possible memory blocks |
|---|---|---|---|
| 0 | 0 | 0 | M0, M2, ..., M28, M30 |
| 1 | 0 | 1 | M0, M2, ..., M28, M30 |
| 2 | 0 | 2 | M0, M2, ..., M28, M30 |
| 3 | 0 | 3 | M0, M2, ..., M28, M30 |
| 4 | 1 | 0 | M1, M3, ..., M29, M31 |
| 5 | 1 | 1 | M1, M3, ..., M29, M31 |
| 6 | 1 | 2 | M1, M3, ..., M29, M31 |
| 7 | 1 | 3 | M1, M3, ..., M29, M31 |

# 2 B.3

**a.**

For the LRU replacement policy, a read hit will access the data subarray, tag, bsubarray and the replacement subarray of the 4 way simultaneously, thus the power consumption is $4 \times (20 + 5 + 1) = 104$ power units.

For the FIFO replacement policy, a read hit will access the data subarray and tag subarray of the 4 way simultaneously, thus the power consumption is

$4 \times (20 + 5) = 100$ power units.

For the random replacement policy, a read hit will access the data subarray and tag subarray of the 4 way simultaneously, thus the power consumption is $4 \times (20 + 5) = 100$ power units.

**b.**

For the LRU replacement policy, a read miss needs an extra memory access compared to the read hit case, thus the power consumption is $104 + 200 = 304$ power units.

For the FIFO replacement policy, it needs to access the memory as well, plus one acess to the replacement subarray, so the power consumption is: $100 + 200 + 1 = 301$ power units.

For the random replacement policy, it is also a memory access plus one read hit: $100 + 200 = 300$ power units.

**c.**

For the LRU replacement policy, a read hit will access the tag subarray and the replacement subarray of the 4 way in the first cycle, and then access the matched data subarray in the second cycle. Thus the power consumption is $4 \times (5 + 1) + 20 = 44$ power units.

For the FIFO replacement policy, a read hit will access the tag subarray of the 4 way in the first cycle, and then access the matched data subarray in the second cycle. Thus the power consumption is $4 \times 5 + 20 = 40$ power units.

For the random replacement policy, a read hit will access the tag subarray of the 4 way in the first cycle, and then access the matched data subarray in the second cycle. Thus the power consumption is $4 \times 5 + 20 = 40$ power units.

**d.**

For the LRU replacement policy, it will access the tag subarray and the replacement subarray of the 4 way in the first cycle, and then access the memory. Thus the power consumption is $4 \times (5 + 1) + 200 = 224$ power units.

For the FIFO replacement policy, it will access the tag subarray of the 4 way in the first cycle, and then one access to memory and one access to the replacement subarray. Thus the power consumption is $4 \times 5 + 200 + 1 = 221$ power units.

For the random replacement policy, it will access the tag subarray of the 4 way in the first cycle, and then one access to memory. Thus the power consumption is $4 \times 5 + 200 = 220$ power units.

**e.**

For the LRU replacement policy, if the way predictor hits, it access the tag subarray, replacement subarray and data subarray for only the predicated way, thus the power consumption is $5 + 1 + 20 = 26$ power units.

For the FIFO replacement policy, it's one access to the tag subarray and the data subarray for only the predicated way, thus the power consumption is $5 + 20 = 25$ power units.

For the random replacement policy, it the same as the FIFO, thus 25 power units.

**f.**

Way predictor miss followed by a two-cycle cache read (the cache read split across two cycles) hit is equivalent to a normal read hit plus one extra access to tag subarray.

For the LRU replacement policy, it's $44 + 5 + 1 = 50$ power units.

For the FIFO replacement policy, it's $40 + 5 = 45$ power units.

For the random replacement policy, it's $40 + 5 = 45$ power units.

**g.**

Way predictor miss followed by a two-cycle cache read (the cache read split across two cycles) miss is equivalent to a normal read miss plus one extra cycle 1.

For the LRU replacement policy, it's $224 + 5 + 1 = 230$ power units.

For the FIFO replacement policy, it's $221 + 5 = 226$ power units.

For the random replacement policy, it's $220 + 5 = 225$ power units.

**h.**

$$P(\text{case e}) = P(\text{cache hit}) \times P(\text{way hit}) = 0.97 * 0.95 = 0.9215$$
$$P(\text{case f}) = P(\text{cache hit}) \times P(\text{way miss}) = 0.97 * 0.05 = 0.0485$$
$$P(\text{case g}) = P(\text{cache miss}) = 0.03$$

In the above equation, $P(caseg) = P(cachemiss)$ because a cache miss implies a way miss.

For the LRU replacement policy: $26 \times 0.9215 + 50 \times 0.0485 + 230 \times 0.03 = 33.05$ power units.

For the FIFO replacement policy, it's $25 \times 0.9215 + 45 \times 0.0485 + 226 \times 0.03 = 31.775$ power units.

For the random replacement policy, it's $25 \times 0.9215 + 45 \times 0.0485 + 225 \times 0.03 = 31.745$ power units.

# 3 B.5

**a.**

Average memory access time for instrution access = (L1 I-cache miss rate) × (average access time from L1 I-cache to L2 cache) + (L1 I-cache miss rate) × (L2 cache miss rate) × (average access time from L2 cache to the main memory + average L2 cache dirty block write back time)

Average access time from L1 I-cache to L2 cache = $15\text{ns} + \frac{32 \text{ bytes}}{128 \text{ bits}} \times \frac{1 \times 10^9}{266\text{MHz}}\text{ns} = 22.519$ ns

Average access time from L2 cache to main memory = $60\text{ns} + \frac{64 \text{ bytes}}{128 \text{ bits}} \times \frac{1 \times 10^9}{133\text{MHz}}\text{ns} = 90.075$ ns

Average L2 cache dirty block write back time = $0.5 \times 60\text{ns} + \frac{64 \text{ bytes}}{128 \text{ bits}} \times \frac{1 \times 10^9}{133\text{MHz}}\text{ns} = 45.038$ ns

Thus, the average memory access time for instrution access = $0.02 \times 22.519 + 0.02 \times (1 - 0.8) \times (90.075 + 45.038) = 0.991\text{ns}$.

**b.**

Average memory access time for data read = (L1 D-cache miss rate) × (average access time from L1 D-cache to L2 cache) + (L1 D-cache miss rate) × (L2 cache miss rate) × (average access time from L2 cache to the main memory + average L2 cache dirty block write back time)

Average access time from L1 D-cache to L2 cache = $15\text{ns} + \frac{16 \text{ bytes}}{128 \text{ bits}} \times \frac{1 \times 10^9}{266\text{MHz}}\text{ns} = 18.759$ ns

Average access time from L2 cache to main memory = 90.075ns

Average L2 cache dirty block write back time = 45.038ns

Thus, the average memory access time for data read = $0.05 \times 18.759 + 0.05 \times (1 - 0.8) \times (90.075 + 45.038) = 2.289\text{ns}$.

**c.**

Since the write policy for the L1 D-cache is write through, and the write policy for the L2 cache is write back, we suppose that a write to L1 D-cache will also write to L2 cache, and a write to L2 cache will only write to main memory if

the we miss the block in L2 cache. We also suppose that if we miss a block when we write to cache (both L1 and L2), we will load that block into cache. And because the L1 D-cache is write through, there is no dirty block in the L1 D-cache.

Average memory access time for data write = (L1 D-cache hit rate) ×0.05×(average data write time from L1 D-cache to L2 cache) + (L1 D-cache miss rate) × (0.05×average data write time from L1 D-cache to L2 cache + average memory access time for data read)

Average data write time from L1 D-cache to L2 cache = (L2 cache hit rate)×(average access time from L1 D-cache to L2 cache)+(L2 cache miss rate)×(average data write time from L2 cache to main memory + average data read time from L2 cache to main memory + average L2 cache dirty block write back time)

Average access time from L1 D-cache to L2 cache = 18.759ns.

Average data write time from L2 cache to main memory = average access time from L2 cache to main memory = 90.075ns.

Average data read time from L2 cache to main memory = average access time from L2 cache to main memory = 90.075ns.

Average L2 cache dirty block write back time = 45.038ns.

Average data write time from L1 D-cache to L2 cache = 0.8×18.759+0.2×(90.075 + 90.075 + 45.038) = 60.045ns.

Average memory access time for data read = 2.289ns.

Average memory access time for data write = 0.95×0.05×60.045 + 0.05×0.05×(60.045 + 2.289) = 3.008ns.

**d.**

CPU clock cycle = $\frac{1}{1.1\ \text{GHz}}$ = 0.909ns.

instruction access = $\frac{0.991}{0.909}$ = 1.090 cycle.

data read = $\frac{2.289}{0.909}$ = 2.518 cycle.

data write = $\frac{3.008}{0.909}$ = 3.309 cycle.

The overall CPI = 1.35+1.09+0.2×2.518+0.1×3.309 = 3.275

# 4 B.8

**a.**

4+4×(miss penalty)+(write cost) = 504 cycles.

**b.**

1 16-byte cache line could hold 4 array elements, and the cache is big enough to hold all the 4 arrays, so we will have 3 read miss and 1 write miss in every 4 iterations, and we don't need to consider evicting blocks in the cache to hold new blocks. The average iteration cost = $\frac{1}{4}\times 504 + \frac{3}{4}\times 4 = 129$ cycles.

**c.**

1 64-byte cache line could hold 16 array elements, and the cache is big enough to hold all the 4 arrays, so we will have 3 read miss and 1 write miss in every 16 iterations, and we don't need to consider evicting blocks in the cache to hold new blocks. The average iteration cost = $\frac{1}{16}\times 504 + \frac{15}{16}\times 4 = 35.25$ cycles.

**d.**

Suppose the cache line in this case is 16 bytes, thus cache line can hold 4 array elements. With 16-byte cache line, there will be 2×1024/16=128 sets(blocks). And the total size of the 4 arrays is 512 blocks. All the 4 arrays could be loaded into the cache with $\left(\frac{512}{128} - 1\right)\times 128$=384 replacements. Each replacement need to write the dirty block back to memory, incurring a write cost. Thus the average number of cycle for each iteration = $\frac{1}{4}\times 504 + \frac{3}{4}\times 4 + \frac{384\times 100}{512} = 204$ cycles.

# 5 B.10

**a.**

Access the missed block from L2. If L2 hit, read the block to L1. If L2 miss, read the block from main memory to both L2 and L1. Whenever a block is evicted from L1, store it to L2 if it is not there. If storing the evicted block evicts another block in L2, check if the block being evicted in L2 exists in L1. If it exists, invalidate that block in L1.

**b.**

Access the missed block from L2. If L2 hit, read the block to L1. If L2 miss, read the block from main memory to L1. Whenever a block is evicted from L1, store the evicted block to L2, and whenever a block is read from L2 to L1, invalidate it in L2.

**c.**

For case a, if the L1 evicted block is dirty, it should be written to L2 even if the block is there. For case b, the possibility of the dirty block doesn't matter.