Homework 6

ECE2500 CRN:82943

Jacob Abel

November 28, 2018

Note: Assume all numbers are in decimal unless explicitly preceded by `0x` for hexadecimal and `0b` for binary.

Problem 1: Explain briefly, the principles of spatial and temporal locality in the context of cache design and memory hierarchy.

Temporal locality is the likelihood of a memory address being accessed based on the distance between the program counter and an instruction. High temporal locality occurs when a memory address is accessed repeatedly such as in a loop sequence. As data is repeatedly accessed it is promoted up the cache. Similarly this can be used in tandem with branch prediction to cache items along the instruction path the branch predictor selects.

Spatial locality is the likelihood of a memory address being accessed based on the distance between the address and recently accessed memory locations. High spatial locality occurs when memory addresses are close to each other, i.e. the delta between two addresses. As such, data near the recently accessed memory addresses can be cached and the aggressiveness of this caching can be determined by detecting common access patterns (like stack frames) or by a algorithm similar to a Kalman filter which can be used to predict the path/direction that future memory accesses will take based on prior accesses. These predictors then signal the cache level and addresses of memory to cache.

While temporally local data is not necessarily spatially local, there is a clear benefit to providing spatial locality to temporally local data when designing an executable as data packed within the same cache lines /blocks will require fewer cache fetches and / or cache misses.

Problem 2: Suppose we have a cache that can hold a total of 8 blocks with 4 words per block.

1. How many bytes does one block contain (block size)?

   $\text{sz}_{\text{block}} = 4 \times \text{sz}_{\text{word}}$

   Assuming 32-bit words, $\text{sz}_{\text{block}} = 4 \times 4\text{bytes} = 16\,\text{bytes}$

2. How many bytes does the cache contain (cache size)?

   $\text{sz}_{\text{cache}} = 8 \times \text{sz}_{\text{block}} = 32 \times \text{sz}_{\text{word}}$

   Assuming 32-bit words, $\text{sz}_{\text{cache}} = 8 \times 16\text{bytes} = 128\,\text{bytes}$

3. Suppose that the cache is designed as a direct-mapped cache. Compute the block index, block offset, and the tag for the following addresses:

   (a) `0x00000004`

   Block Index $\text{index} = \left\lfloor \frac{0x00000004}{16} \right\rfloor \bmod 8 = 0x0$
   Block Offset $\text{offset} = 0x00000004 \bmod 16 = 0x4$
   Tag $\text{tag} = \left\lfloor \frac{0x00000004}{128} \right\rfloor = 0x0$

   (b) `0x00000014`

   Block Index $\text{index} = \left\lfloor \frac{0x00000014}{16} \right\rfloor \bmod 8 = 0x1$
   Block Offset $\text{offset} = 0x00000014 \bmod 16 = 0x4$
   Tag $\text{tag} = \left\lfloor \frac{0x00000014}{128} \right\rfloor = 0x0$

   (c) `0x01000004`

   Block Index $\text{index} = \left\lfloor \frac{0x01000004}{16} \right\rfloor \bmod 8 = 0x0$
   Block Offset $\text{offset} = 0x01000004 \bmod 16 = 0x4$
   Tag $\text{tag} = \left\lfloor \frac{0x01000004}{128} \right\rfloor = 0x20000$

   (d) `0x00000008`

   Block Index $\text{index} = \left\lfloor \frac{0x00000008}{16} \right\rfloor \bmod 8 = 0x0$
   Block Offset $\text{offset} = 0x00000008 \bmod 16 = 0x4$
   Tag $\text{tag} = \left\lfloor \frac{0x00000008}{128} \right\rfloor = 0x0$

   (e) `0x00000018`

   Block Index $\text{index} = \left\lfloor \frac{0x00000018}{16} \right\rfloor \bmod 8 = 0x1$
   Block Offset $\text{offset} = 0x00000018 \bmod 16 = 0x8$
   Tag $\text{tag} = \left\lfloor \frac{0x00000018}{128} \right\rfloor = 0x0$

   (f) `0x01000008`

   Block Index $\text{index} = \left\lfloor \frac{0x01000008}{16} \right\rfloor \bmod 8 = 0x0$
   Block Offset $\text{offset} = 0x01000008 \bmod 16 = 0x8$
   Tag $\text{tag} = \left\lfloor \frac{0x01000008}{128} \right\rfloor = 0x20000$

Problem 3: Suppose we have a cache that can hold 16 blocks with 2 words per block. Assume that initially the cache is empty. Draw tables to illustrate the state of the cache after each of the following data read operations from the processor:

The rows in the table should correspond to blocks, with columns for tag, block index, valid, and the memory addresses of data currently stored in the blocks (if no data is stored in a block, simply put dashes). Also, indicate if the processor will stall in order to complete each read operation.

Block Size $\text{sz}_{\text{block}} = 2 \times 4\text{bytes} = 8\,\text{bytes}$

Cache Size $\text{sz}_{\text{cache}} = 16 \times 8\text{bytes} = 128\,\text{bytes}$

Block Index $\text{index} = \left\lfloor \frac{\text{addr}}{8} \right\rfloor \bmod 16$

Tag $\text{tag} = \left\lfloor \frac{\text{addr}}{128} \right\rfloor$

1. Read word from `0x00000000`: CPU will stall

Index	Tag	Valid	Address
0x0	0x000000	1	0x00000000 - 0x00000007
0x1	—	—	—
0x2	—	—	—
0x3	—	—	—
0x4	—	—	—
0x5	—	—	—
0x6	—	—	—
0x7	—	—	—
0x8	—	—	—
0x9	—	—	—
0xA	—	—	—
0xB	—	—	—
0xC	—	—	—
0xD	—	—	—
0xE	—	—	—
0xF	—	—	—

2. Read word from `0x00000004`: CPU will not stall

Index	Tag	Valid	Address
0x0	0x000000	1	0x00000000 - 0x00000007
0x1	—	—	—
0x2	—	—	—
0x3	—	—	—
0x4	—	—	—
0x5	—	—	—
0x6	—	—	—
0x7	—	—	—
0x8	—	—	—
0x9	—	—	—
0xA	—	—	—
0xB	—	—	—
0xC	—	—	—
0xD	—	—	—
0xE	—	—	—
0xF	—	—	—

3. Read word from `0x00000040`: CPU will stall

Index	Tag	Valid	Address
0x0	0x000000	1	0x00000000 - 0x00000007
0x1	—	—	—
0x2	—	—	—
0x3	—	—	—
0x4	—	—	—
0x5	—	—	—
0x6	—	—	—
0x7	—	—	—
0x8	0x000000	1	0x00000040 - 0x00000047
0x9	—	—	—
0xA	—	—	—
0xB	—	—	—
0xC	—	—	—
0xD	—	—	—
0xE	—	—	—
0xF	—	—	—

4. Read word from `0x00000020`: CPU will stall

Index	Tag	Valid	Address
0x0	0x000000	1	0x00000000 - 0x00000007
0x1	—	—	—
0x2	—	—	—
0x3	—	—	—
0x4	0x000000	1	0x00000020 - 0x00000027
0x5	—	—	—
0x6	—	—	—
0x7	—	—	—
0x8	0x000000	1	0x00000040 - 0x00000047
0x9	—	—	—
0xA	—	—	—
0xB	—	—	—
0xC	—	—	—
0xD	—	—	—
0xE	—	—	—
0xF	—	—	—