Jordan Wu

U0900517

Homework Assignment 9

1. Do problems 5.3.1 through 5.3.5 from the book. Show your math and explain your reasoning.

   For clarification, assume memory is byte addressed, that each row holds only one entry (one block of data), and that each row needs store a valid bit, a tag, and storage for the data block. Also, the hit ratio is the percentage given by the number of cache hits / total memory accesses.

   Finally, after you answer 5.3.3, explain why you do not need to store the index (the row number) in the cache.

5.3

For a direct-mapped cache design with a 32-bit address, the following bits of the address are used to access the cache.

Tag	Index	Offset
31-10	9-5	4-0

5.3.1

What is the cache block size (in words)?

Cache represent the level of the memory hierarchy between the processor and main memory. The cache memory hierarchy is a way to improve performance by using the principle of locality. The principle of locality say that saving data in memory into the upper level (closer to the processor) based on recently usage. The cache contains a collection of recent references. One simple cache structure is the direct mapped, assigns a location in the cache for each word in memory is to assign the cache location based on the address of the word in memory.

$$(Block\ address)modulo(Number\ of\ blocks\ in\ the\ cache)$$

The number of blocks in the cache is a power of 2, then modulo can be computed by using the low-order $\log_2$ (cache size in blocks) bits of the address. Tag contain the address information required to identify whether a word in the cache corresponds to the requested word from the process. The cache index is used to select the block in the cache and depends on the number of blocks in the cache. The block address is n-bits fields and has 2^n values. This mean that the index of the block can be found by using the $\log_2(n)$. Remember that memory uses byte addressing (multiples of four bytes), the LSB two bits of very address can be ignored.

- 32-bit addresses
- A direct-mapped cache
- The cache size is 2^n blocks, n bits are used for the index
- The block size is 2^m words (2^{m+2} bytes), m bits are used for the word within the block and two bits are used for the byte part of the address

The following table above will give us the index which is used to find the cache block size

$$index = 5\ bits\ offset$$

$$cache\ block\ size = 2^5\ bytes$$

Now converting this into words, one word in this case is 32 bits and bytes is 8 bits. We know that 32 divided by 8 is equal to 4 which is the same as 2^2.

$$cache\ block\ size = 2^5\ bytes = 2^{m+2}\ bytes$$

Solving for m will give the block size in words, m = 3

$$cache\ block\ size = 2^3 words = 8\ words$$

5.3.2

How many entries does the cache have?

This is asking for the number blocks in the cache (cache size) which depends on the index. The index is 5 bits offset

$$number\ of\ entries = 2^{index} = 2^5 = 32\ entries\ or\ blocks$$

5.3.3

What is the ratio between total bits required for such a cache implementation over the data storage bits?

The total number of bits in a direct-mapped cache is using the following parameter m = 3 and n = index = 5.

$$(cache\ size) \cdot (block\ size + tag\ size + valid\ field\ size)$$

Where,

$$cache\ size = 2^n = 32\ bits$$

$$block\ size = 8\ words = 8 \cdot 32\ bits = 256\ bits$$

$$tag\ size = 32 - (n + m + 2) = 32 - (5 + 3 + 2) = 22\ bits$$

$$valid\ field\ size = 1$$

The total number of bits in a direct-mapped cache is 8928 bits. Since this is a ratio question we must find the total number of bits for 8 words times the cache size as the data storage bits. Data storage bits is equal to 8192.

$$ratio = \frac{(cache\ size) \cdot (block\ size + tag\ size + valid\ field\ size)}{(cache\ size) \cdot (block\ size)} = \frac{8928}{8192} \approx 1.09$$

Since the index field of the address selects the block, architects omit the index bits because they are redundant and since the index field of any address of a cache block must be that block number.

5.3.4

Starting from power on, the following byte-address cache references are recorded.

Address											
0	4	16	132	232	160	1024	30	140	3100	180	2180

How many blocks are replaced?

Replaced mean that if the bit is valid but the tag does not match, the data in that block will be replaced with new data. Here is a table of what is happening using the same format in the textbook.

Tag	Index	Offset
22 bits	5 bits	5 bits

$$modulo = 32\ or\ look\ at\ the\ LSB\ 5\ bits\ of\ the\ address$$

Decimal address of reference	Binary address of reference	Hit or miss in cache	Assigned cache block index	Replacement
0	00000000000000000000000000000000$_{two}$	miss	0	no
4	00000000000000000000000000000100$_{two}$	miss	4	no
16	00000000000000000000000000010000$_{two}$	miss	16	no
132	00000000000000000000000010000100$_{two}$	hit	4	yes
232	00000000000000000000000011101000$_{two}$	miss	8	no
160	00000000000000000000000010100000$_{two}$	hit	0	yes
1024	00000000000000000000010000000000$_{two}$	hit	0	yes
30	00000000000000000000000000011110$_{two}$	miss	30	no
140	00000000000000000000000010001100$_{two}$	miss	12	no
3100	00000000000000000000110000011100$_{two}$	miss	28	no
180	00000000000000000000000010110100$_{two}$	miss	20	no
2180	00000000000000000000100010000100$_{two}$	hit	4	yes

There were four replacement

5.3.5

What is the hit ratio?

Looking at the table above, there were 4 hits out of 12 possible hits/miss. The hit ratio will be 4/12 = 0.333

2. **Assume you have a four-way set-associative cache with enough storage to cache 512 bytes of memory. Assume the block size for each entry is 16 bytes, and an LRU replacement polity is used within each set. Answer these questions, show your math, and explain your reasoning:**

- **How many rows are there in this cache?**

In a set-associative cache there are a fixed number of location where each block can be placed. With n locations for a block is called an n-way set associative cache, this consist of a number of sets, each of which consists of n blocks. Each block in the memory maps to a unique set in the cache given by the index field, and a block can be placed in any element of that set.

Given that the cache's storage is 512 bytes of memory and the block size is 16 bytes. We must find the number of sets using the given information. We know that for the four-way set associative, each set will have 4 blocks. Each block size is 16 bytes, so that each set size is 4 multiply by 16 bytes equals 64 bytes. Now we can divide the storage size 512 bytes with 64 bytes. This will give the value of 8 sets which represents the rows in this cache and like the table below.

Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								
2								
3								
4								
5								
6								
7								

- **Which bits of the memory address serve as the offset, row index, and tag?**

$$offset = log_2(Data\ block\ size\ in\ bytes) = log_2(16\ bytes) = 4\ bits$$

$$row\ index = log_2(Number\ of\ sets) = log_2(8) = 3\ bits$$

$$tag = 32 - offset\ bits - row\ index\ bits = 25\ bits$$

Tag	Index	Block offset
25 bits	3 bits	4 bits

- **Assume a memory access is made and that it misses our set-associative cache. Once the memory access finishes accessing RAM, the data will be cached. Describe how a four-way set-associative cache determines *where* the data should be placed within the cache.**

Set-associative cache usually replace the least recently used block within a set; that is, the block that was used furthest in the past if replaced. The most commonly used scheme is least recently used (LRU), this scheme the block replaced is the one that has been unused for the longest time. This is done by implementing a bit in each set and setting the bit to indicate whenever that element is referenced.

3. **Do problems 5.5.1 through 5.5.3 from the book. Show your math and explain your reasoning.**

   For problem 5.5.1, ignore the term 'working set' (but answer the rest of the question). Also, the 3C's are defined on page 459.

5.5

Media application that play audio or video files are part of a class of workloads called "streaming" workloads; i.e., they bring in large amounts of data but do not reuse much of it. Consider a video streaming workload that accesses a 512 KiB working set sequentially with the following address stream:

0, 2, 4, 6, 8, 10, 12, 14, 16…

5.5.1

Assume a 64 KiB direct-mapped cache with a 32-byte block. What is the miss rate for the address stream above? How is this miss rate sensitive to the size of the cache or the working set? How would you categorize the misses this world load is experiencing, based on the 3C model?

1 kibibyte is equal to 1024 bytes, this mean that for 64 KiB = 64(1024) = 65536 bytes of storage. Each block is 32 bytes and is a direct-mapped cache. We can find the number of blocks/rows in this cache by dividing the storage by the block size.

$$cache\ size = \frac{65536\ bytes}{32\ bytes} = 2048\ blocks/rows$$

The sequence is just the multiples of 2

$$address\ stream = \sum_{i=0}^{M} 2 \cdot i$$

$$blocks\ in\ \text{“streaming” workloads} = \frac{512\ KiB}{32\ bytes} = \frac{524288\ bytes}{32\ bytes} = 16384\ blocks$$

In the direct-mapped cache the modulo is $\log_2(2048) = 11$ bits in index. The first time it is being access will always miss, since there is no data in the cache. Only the even index are being accessed from index 0 to 2048. After missing everything the first time and storing the data into the data block for each even row, everything will hit after the sequence 2048. This is because the same rows are being accessed again and assuming the tag matches. The total number of time we loop through the even index is $16384\ blocks/2048\ blocks = 8$. This means the miss ratio is given by

$$miss\ rate = \frac{miss\ when\ initial\ state\ of\ cache\ after\ power - on}{Number\ of\ loops\ through\ same\ rows} = \frac{1}{8} = .125$$

Compulsory misses: These are the cache that misses when first accessing the blocks in the cache, also known as cold-start misses.

5.5.2

Re-compute the miss rate when the cache block size is 16 bytes, 64 bytes, and 128 bytes. What kind of locality is this workload exploiting?

When changing the block size to a different value, this will change the rows in the cache.

For block size 16 bytes:

$$cache\ size = \frac{65536\ bytes}{16\ bytes} = 4096\ blocks/rows$$

This increased our rows in the cache and change how many times we loop through the cache. The total number of time we loop through the even index is $16384 blocks/4096\ blocks = 4$

$$miss\ rate = \frac{miss\ when\ initial\ state\ of\ cache\ after\ power - on}{Number\ of\ loops\ through\ same\ rows} = \frac{1}{4} = .25$$

For block size 64 bytes:

$$cache\ size = \frac{65536\ bytes}{64\ bytes} = 1024\ blocks/rows$$

The total number of time we loop through the even index is $16384 blocks/1024\ \ blocks = 16$

$$miss\ rate = \frac{miss\ when\ initial\ state\ of\ cache\ after\ power - on}{Number\ of\ loops\ through\ same\ rows} = \frac{1}{16} = .0625$$

For block size 128 bytes:

$$cache\ size = \frac{65536\ bytes}{128\ bytes} = 512\ blocks/rows$$

The total number of time we loop through the even index is $16384 blocks/512\ blocks$ = 32

$$miss\ rate = \frac{miss\ when\ initial\ state\ of\ cache\ after\ power - on}{Number\ of\ loops\ through\ same\ rows} = \frac{1}{32} = .03125$$

Seem like the more row we have in our cache the higher the miss rate is, due to the cold-start misses. This is exploiting the Temporal locality, if something is referenced, it might be referenced again soon.

5.5.3

"Prefetching" is a technique that leverages predictable address patterns to speculatively bring in additional cache blocks when a particular cache block is accessed. One example of prefetching is a stream buffer that prefetching sequentially adjacent cache blocks into a separate buffer when a particular cache block is brought in. If the data is found in the prefetch buffer, it is considered as a hit and moved into the cache and the next cache block is prefetched. Assume a two-entry stream buffer and assume that the cache latency is such that a cache block can be loaded before the computation on the previous cache block is completed. What is the miss rate for the address stream above?

This says that whenever the cache block is accessed it will prefetch the next sequentially block into a separate buffer. If that data is found in the prefetch buffer stated in the previous sentence then it is consider a hit and is moved into the cache and the next block is prefetched. I am assume that it will miss once and any time after that it will hit since sequentially cache block is prefetched into the separate buffer. The miss rate should be zero if it misses once and hit every time after.