5

Monday, April 12, 2021 1:13 AM

1. Assume that the system uses 32-bit address.
   a. Consider a cache with 64MB of data, 4-way set associativity, and 128-byte line (block) size. What is the size of each line's tag in bits?

- Cache has 64MB of data, or 67,108,864 bytes.
- Each block is 128 bytes, so 67,108,864/128 = 524,288 blocks
- Since we have 4-way S.A. then each set has 4 blocks.
  - Therefore we have to index on sets, so: 524,288/4 = 131,072 sets.
  - Index field needs
    $2^n = 131,072$
    $n = 17$ ∴ 17 bits required to distinguish between all sets.

- Each block has 128 bytes. Block offset field then needs
  $2^n = 128$
  $n = 7$, 7 bits needed to distinguish between all bytes in a block.

$\boxed{\text{Tag size} = 32 - 17 - 7 = 8}$

   b. Consider a cache with 2048 sets, 4-way set associativity, and 12-bit tags. How many bytes can the cache store? (What is the data capacity not including overhead of storing tags, valid bits, etc.?)

? ? ? ? ? ?

- 4-way set associativity, ∴ each set has 4 blocks.
- We have 2048 sets, so in total we have:
  $(2048 \text{ sets}) \times (4 \frac{\text{blocks}}{\text{set}}) = 8192$ total blocks.
- We have 12 bit tags.
  $32 - 12 = 20$ bits available between block offset & index field value.

- How many bytes per block? (what is block size?)

- How many bits in index field?
  ↳ We have to distinguish between sets, so we have $2^n = 2048$, $n = 11$
  ∴ $x + 12 + 11 = 32$
  $x = 9$ bits required to distinguish bytes in a block
  ∴ Every block has $2^9 = 512$ bytes.
  ∴ $\boxed{\text{Cache Size} = 2048 \times 4 \times 512 = 4,194,304}$

   c. Consider a cache with a 256-byte line size and 14-bit tags. The cache has 4096 lines. What is the set associativity of the cache?

- Block size is 256 bytes.
  ∴ We need $2^n = 256$, $n = 8$ bits required to distinguish bytes in a block.

- Tag is 14 bit so $32 - 14 - 8 = 10$ bits required to distinguish sets
  ∴ we have 2^{10} sets or 1024.

- 4096 total blocks, and 1024 sets:

$\boxed{\frac{4096 \text{ blocks}}{1024 \text{ sets}} = 4\text{-way set associativity}}$

2. In this problem, we compare cache replacement policies for a given trace of accesses to a set. A trace is made of consecutive block addresses dynamically accessed by a program. Take the following trace, where each letter is a block address:

abcdcbaefgfaefgdcbaefga

Assume that the fully associative cache that has four cache lines is cold (empty) at the beginning. What is the miss rate under the following replacement policies:

- FIFO
- LRU

Accesses	a	b	c	d	c	b	a	e	f	f	g	f	a	e	f	g	d	c	b	a	e	f	g	a
priority order (LRU) 0 (MRU)	a	b	c	d	c	b	a	e	f	f	g	f	a	e	f	g	d	c	b	a	e	f	g	a
1		a	b	c	d	c	b	a	e	e	f	g	f	a	e	f	g	d	c	b	a	e	f	g
2			a	b	b	d	c	b	a	a	e	e	g	f	a	e	f	g	d	c	b	a	e	f
3 (LRU)				a	a	a	d	c	b	b	a	a	e	g	g	a	e	f	g	d	c	b	a	e
cache miss	M	M	M	M	H	H	H	M	M	H	M	H	H	H	H	H	H	M	M	M	M	M	M	H

$\boxed{\text{Hit rate: } 10/24 \approx 42\% \quad \text{Miss rate: } 14/24 \approx 58\%}$

Accesses	a	b	c	d	c	b	a	e	f	f	g	f	a	e	f	g	d	c	b	a	e	f	g	a
priority order (FIFO) 0 oldest	a	b	c	d	d	d	d	e	f	f	g	g	a	a	a	d	c	b	b	c	f	g	a	
1 oldest		a	b	c	c	c	c	d	e	e	f	f	g	g	g	a	d	c	c	b	e	f	g	
2 oldest			a	b	b	b	b	c	d	d	e	e	f	f	f	g	a	d	d	c	b	t	f	
3 oldest				a	a	a	a	b	c	c	d	d	e	e	e	f	g	a	a	d	c	b	e	
cache miss	M	M	M	M	H	H	H	M	M	H	M	H	M	H	H	M	M	M	H	M	M	M	M	M

$\boxed{\text{Hit rate: } 9/24 = 37.5\% \quad \text{Miss rate: } 15/24 = 62.5\%}$

3. In this problem, you will evaluate cache performance with your MatrixMul program that was implemented in HW2. Use any MatrixMul code. If you couldn't finish the MatrixMul implementation in HW2, you can use other students' code. We won't check plagiarism for the code itself. But, the evaluation should be done by yourself. This is NOT a group work. You can use MARS tool to collect cache hits/misses.

   a. Overwrite the input and output matrixes in your original MatrixMul code with the following matrixes.

   b. You can measure cache performance by using MARS Data Cache simulator plugin. Click "Tools->Data Cache Simulator" in the MARS menu bar. Then, the cache simulator tool will appear like below:

The configurable parameters for the cache include Placement Policy (a.k.a. mapping scheme), Block Replacement Policy (always leave at LRU), Set size (a.k.a. K-Ways), Number of blocks in cache and cache block size. The total cache size (in bytes) is simply calculated as: (# of Blocks) * (Cache block size in words) * (4 bytes per word). Note: There is an oddity that statistics are not kept for any cache with 512 blocks or more even though those options exist in the drop down box.

To run data cache tool, click "Connect to MIPS" button of the Data Cache tool so the tool can begin monitoring the MARS simulation. **Everytime you run your program, you should also be sure to RESET the cache simulator** by clicking the Reset button.

To configure the cache, run the following experiments on your baseline code.

Cache misses can be categorized as compulsory, capacity, or conflict misses. Determine the number of compulsory misses. To do this, set your block size at **4 words per block** and set **replacement policy to LRU**. Think about what mapping scheme should be used if you do not want any conflict misses. Then continue to increase the size of the cache **by increasing the number of blocks** until you can correctly determine the number of compulsory misses. Find the smallest cache size that only produces compulsory misses.

→ To eliminate conflict misses, we must set to fully associative mapping

- Threshold value for cache miss:
- Compulsory misscount: 65
- Percentage: ~3%

   c.

1.

cache size	cache hit rate	cache miss count
(50%) 32	95%	107
(25%) 16	87%	302

2. Change the placement policy to the other two types (e.g., if you set direct mapping in step b, use fully set associative and N-way set associative; feel free to set the N size as you wish) and find the configuration that provide better performance than the results in step c.1. (e.g., for my MatrixMul code, 4-way set associative cache shows more cache hits than direct-mapped cache that was used for determining the cache size)

2. Change the placement policy to the other two types (e.g., if you set direct mapping in step b, use fully set associative and N-way set associative; feel free to set the N size as you wish) and find the configuration that provide better performance than the results in step c.1. (e.g., for my MatrixMul code, 4-way set associative cache shows more cache hits than direct-mapped cache that was used for determining the cache size)

→ Changing Placement Policy to 4-way associative.

Mapping: 4-way set associative
Block RP: LRU
Number of blocks: 64

$\boxed{\text{Cache hit rate: } 99\%}$

   d. **In the homework submission, include the answers of the following questions based on your evaluations above.**

1. What placement policy did you set to find the cache size in step b? → Fully Associative
2. What is the cache size that has compulsory misses only (the size that you found in step b)? → 64 blocks. Each block has 4 words, or 16 bytes. We have 64 blocks × 16 bytes/block = 1024 bytes size cache
3. How many compulsory misses did you have? → 65
4. Put a table or a graph that shows "cache miss count" and "cache hit rate" when you decrease cache size to 50%, 25%, and 12.5%.

cache size	cache hit rate	cache miss count
(50%) 32	95%	107
(25%) 16	87%	302
(12.5%) 8	55%	1114

5. Explain what cache configuration showed better cache hit rate when the cache size is decreased to 50%, 25%, and 12.5%. Show the cache miss count and hit rate of the configuration for each cache size. And, add your reasoning why the configurations show better cache hit rates by emphasizing the memory access pattern of the MatrixMul code.

Using this configuration:

cache size	cache hit rate	cache miss count
(50%) 32	99%	33
(25%) 16	94%	142
(12.5%) 8	62%	892

The cache configuration of the three listed in the table that was best was the one using 32 as the number of blocks. Above all the other ones it showed the best cache hit rate, and least cache miss count.

By increasing the cache block size (words) we were able to get better results than using the configuration with fully associative mapping because each block can contain more data.