CS152 Computer Architecture and Engineering
CS252 Graduate Computer Architecture
Spring 2019

Caches and the Memory Hierarchy

Assigned February 13 Problem Set #2 *Due Wed, February 27*

http://inst.eecs.berkeley.edu/~cs152/sp19

The problem sets are intended to help you learn the material, and we encourage you to collaborate with other students and to ask questions in discussion sections and office hours to understand the problems. However, each student must turn in his/her own solution to the problems.

The problem sets also provide essential background material for the exams. The problem sets will be graded primarily on an effort basis, but if you do not work through the problem sets you are unlikely to succeed at the exams! Homework assignments are due at the beginning of class on the due date. Late homework will not be accepted.

Problem 1: Cache Access-Time & Performance

This problem requires the knowledge of Handout #2 (Cache Implementations at http://www-inst.eecs.berkeley.edu/~cs152/sp19/handouts/sp19/handout2.pdf) and Lectures 6 & 7. Please, read these materials before answering the following questions.

Ben is trying to determine the best cache configuration for a new processor. He knows how to build two kinds of caches: direct-mapped caches and 4-way set-associative caches. The goal is to find the better cache configuration with the given building blocks. He wants to know how these two different configurations affect the clock speed and the cache miss-rate, and choose the one that provides better performance in terms of average latency for a load.

Problem 1.A — Access Time: Direct-Mapped

Now we want to compute the access time of a direct-mapped cache. We use the implementation shown in Figure H2-A in Handout #2. Assume a 128-KB cache with 8-word (32-byte) cache lines. The address is 32 bits and byte-addressed, so the two least significant bits of the address are ignored since a cache access is word-aligned. The data output is also 32 bits (1 word), and the MUX selects one word out of the eight words in a cache line. Using the delay equations given in Table 2.1-1, **fill in the column for the direct-mapped (DM) cache in the table**. *In the equation for the data output driver, 'associativity' refers to the associativity of the cache (1 for direct-mapped caches, A for A-way set-associative caches).*

Component	Delay equation (ps)		DM (ps)	SA (ps)
Decoder	20×(# of index bits) + 100	Tag		
		Data		
Memory array	$20\times \log_2$ (# of rows) + $20\times \log_2$ (# of bits in a row) + 100	Tag		
		Data		
Comparator	20×(# of tag bits) + 100			
N-to-1 MUX	$50\times\log_2 N + 100$			
Buffer driver	200			
Data output driver	50×(associativity) + 100			
Valid output driver	100			

Table 2.1-1: Delay of each Cache Component

What is the critical path of this direct-mapped cache for a cache read? What is the access time of the cache (the delay of the critical path)? To compute the access time, assume that a 2-input gate (AND, OR) delay is 50 ps. If the CPU clock is 1.5 GHz, how many CPU cycles does a cache access take?

Problem 1.B Access Time: Set-Associative

We also want to investigate the access time of a set-associative cache using the 4-way set-associative cache in Figure H2-B in Handout #2. Assume the total cache size is still 128-KB (each way is 32-byes), a 4-input gate delay is 100 ps, and all other parameters (such as the input address, cache line, etc.) are the same as part 2.1.A. **Compute the delay of each component, and fill in the column for a 4-way set-associative cache in Table 2.1-1.**

What is the critical path of the 4-way set-associative cache? What is the access time of the cache (the delay of the critical path)? What is the main reason that the 4-way set-associative cache is slower than the direct-mapped cache? If the CPU clock is 1.5 GHz, how many CPU cycles does a cache access take?

Problem 1.C Miss-rate analysis

Now Ben is studying the effect of set-associativity on the cache performance. Since he now knows the access time of each configuration, he wants to know the miss-rate of each one. For the miss-rate analysis, Ben is considering two small caches: a direct-mapped cache with 8 lines with 16 bytes/line, and a 4-way set-associative cache of the same size and line size. For the set-associative cache, Ben tries out two replacement policies – least recently used (LRU) and round robin (FIFO).

Ben tests the cache by accessing the following sequence of hexadecimal byte addresses, starting with empty caches. For simplicity, assume that the addresses are only 12 bits. **Complete the following tables** for the direct-mapped cache and both types of 4-way set-associative caches showing the progression of cache contents as accesses occur (in the tables, 'inv' = invalid, and the column of a particular cache line contains the tag of that line). Also, for each address calculate the tag and index (which should help in filling out the table). *You only need to fill in elements in the table when a value changes.*

D-map	**Addresses and tags are in HEX**								
	line in cache (tag)								hit?
Address	L0	L1	L2	L3	L4	L5	L6	L7	
110	inv	2	inv	inv	inv	inv	inv	inv	no
136				2					no
202	4								no
1A3									
102									
361									
204									
114									
1A4									
177									
301									
206									
135									

	D-map
Total Misses	
Total Accesses	

4-way Address	**LRU -- addresses and tags are in HEX**								
	line in cache								hit?
	Set 0				Set 1				
	way0	**way1**	**Way2**	**way3**	**way0**	**way1**	**way2**	**way3**	
110	Inv	Inv	Inv	inv	8	inv	inv	inv	no
136						9			no
202	10								no
1A3									
102									
361									
204									
114									
1A4									
177									
301									
206									
135									

	4-way LRU
Total Misses	
Total Accesses	

4-way Address	**FIFO -- addresses and tags are in HEX**								
	line in cache (tag)								hit?
	Set 0				Set 1				
	way0	**way1**	**way2**	**way3**	**way0**	**way1**	**way2**	**way3**	
110	inv	inv	inv	inv	8	inv	inv	inv	no
136						9			no
202	10								no
1A3									
102									
361									
204									
114									
1A4									
177									
301									
206									
135									

	4-way FIFO
Total Misses	
Total Accesses	

Problem 1.D Average Latency

Assume that the results of the above analysis can represent the average miss-rates of the direct-mapped and the 4-way set-associative 128-KB caches studied in 1.A and 1.B. What would be the average memory access latency in CPU cycles for each cache (assume that the cache miss penalty is 20 cycles)? Which one is better? For the different replacement policies for the set-associative cache, which one has a smaller cache miss rate for the address stream in 1.C? Explain why. Is that replacement policy always going to yield better miss rates? If not, give a counter example using an address stream.

Problem 2: Loop Ordering

This problem requires knowledge of Lecture 7. Please, read it before answering the following questions.

This problem evaluates the cache performances for different loop orderings. You are asked to consider the following two loops, written in C, which calculate the sum of the entries in a 128 by 32 matrix of 32-bit integers:

Loop A	***Loop B***
`sum = 0;`	`sum = 0;`
`for (i = 0; i < 128; i++)`	`for (j = 0; j < 32; j++)`
`  for (j = 0; j < 32; j++)`	`  for (i = 0; i < 128; i++)`
`    sum += A[i][j];`	`    sum += A[i][j];`

The matrix `A` is stored contiguously in memory in row-major order. Row major order means that elements in the same row of the matrix are adjacent in memory as shown in the following memory layout:

`A[i][j]` resides in memory location `[4*(32*i + j)]`

Memory Location:

0	4		124	128		4*(32*127+31)
A[0][0]	A[0][1]	...	A[0][31]	A[1][0]	...	A[127][31]

For *Problem 2.A* to *Problem 2.C*, assume that the caches are initially empty. Also, assume that only accesses to matrix `A` cause memory references and all other necessary variables are stored in registers. Instructions are in a separate instruction cache.

Problem 2.A

Consider a 4KB direct-mapped data cache with 8-word (32-byte) cache lines.
Calculate the number of cache misses that will occur when running Loop A.
Calculate the number of cache misses that will occur when running Loop B.

The number of cache misses for Loop A:____________________________

The number of cache misses for Loop B:____________________________

Problem 2.B

Consider a direct-mapped data cache with 8-word (32-byte) cache lines. Calculate the minimum number of cache lines required for the data cache if Loop A is to run without any cache misses other than compulsory misses. Calculate the minimum number of cache lines required for the data cache if Loop B is to run without any cache misses other than compulsory misses.

Data-cache size required for Loop A: ____________________________ cache line(s)

Data-cache size required for Loop B: ____________________________ cache line(s)

Problem 2.C

Consider a 4KB fully-associative data cache with 8-word (32-byte) cache lines. This data cache uses a first-in/first-out (FIFO) replacement policy.
Calculate the number of cache misses that will occur when running Loop A.
Calculate the number of cache misses that will occur when running Loop B.

The number of cache misses for Loop A:____________________________

The number of cache misses for Loop B:____________________________

Problem 3: Microtagged Cache

In this problem, we explore *microtagging*, a technique to reduce the access time of set-associative caches. Recall that for associative caches, the tag check must be completed before load results are returned to the CPU, because the result of the tag check determines which cache way is selected. Consequently, the tag check is often on the critical path.

The time to perform the tag check (and, thus, way selection) is determined in large part by the size of the tag. We can speed up way selection by checking only a subset of the tag—called a microtag—and using the results of this comparison to select the appropriate cache way. Of course, the full tag check must also occur to determine if the cache access is a hit or a miss, but this comparison proceeds in parallel with way selection. We store the full tags separately from the microtag array.

We will consider the impact of microtagging on a 4-way set-associative 16KB data cache with 32-byte lines. Addresses are 32 bits long. Microtags are 8 bits long. The baseline cache (i.e. without microtagging) is depicted in Figure H2-B in the handout 2. Figure 1, below, shows the modified tag comparison and driver hardware in the microtagged cache.

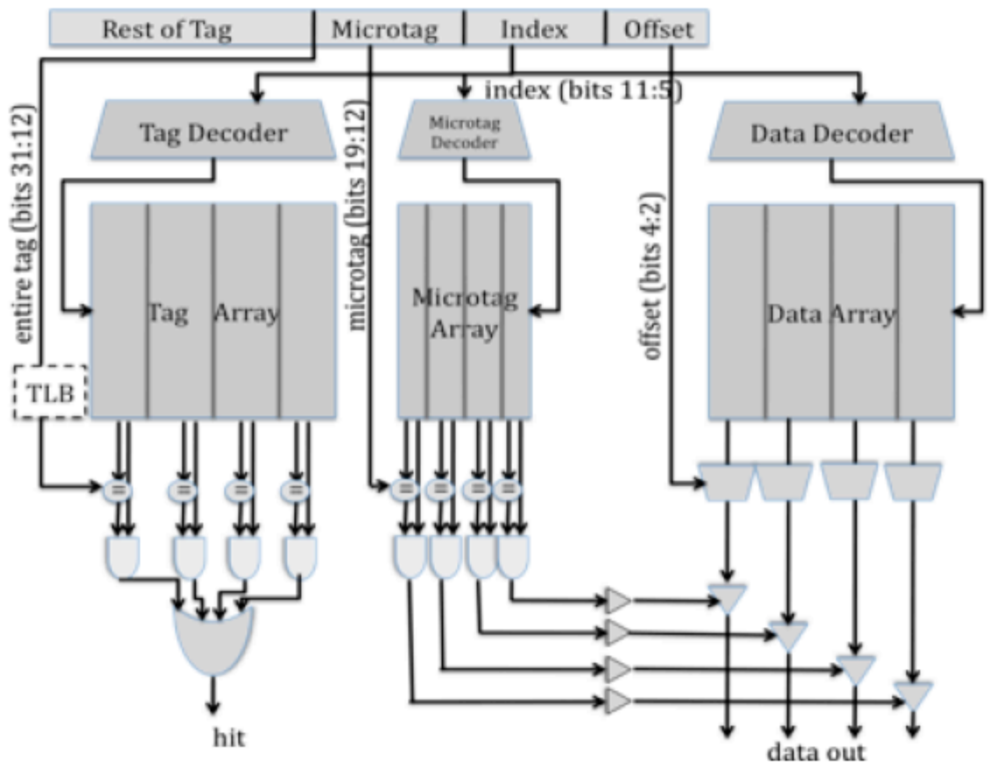


Figure 2.4-1: Microtagged cache datapath

Problem 3.A **Cache Cycle Time**

Table 2.4-1, below, contains the delays of the components within the 4-way set-associative cache, for both the baseline and the microtagged cache. For both configurations, determine the critical path and the cache access time (i.e., the delay through the critical path). **Fill in the columns for the microtagged cache.**

Assume that the 2-input AND gates have a 50ps delay and the 4-input OR gate has a 100ps delay.

Component	Delay equation (ps)		Baseline	Microtagged
Decoder	20×(# of index bits) + 100	Tag	240	
		Data	240	
Memory array	$20 \times \log_2$ (# of rows) + $20 \times \log_2$ (# of bits in a row) + 100	Tag	369	
		Data	440	
		Microtag		
Comparator	20×(# of tag bits) + 100	Tag	500	
		Microtag		
N-to-1 MUX	$50 \times \log_2 N + 100$		250	250
Buffer driver	200		200	200
Data output driver	50×(associativity) + 100		300	300
Valid output driver	100		100	100

Table 2.4-1: Delay of each Cache Component

What is the old critical path? The old cycle time (in ps)?

What is the new critical path? The new cycle time (in ps)?

Problem 3.B AMAT

Assume temporarily that both the baseline cache and the microtagged cache have the same hit rate, 95%, and the same average miss penalty, 20 ns. Using the cycle times computed in 3.A as the hit times, compute the average memory access time for both caches. **What was the old AMAT (in ns)? What is the new AMAT (in ns)?**

Problem 3.C **Constraints**

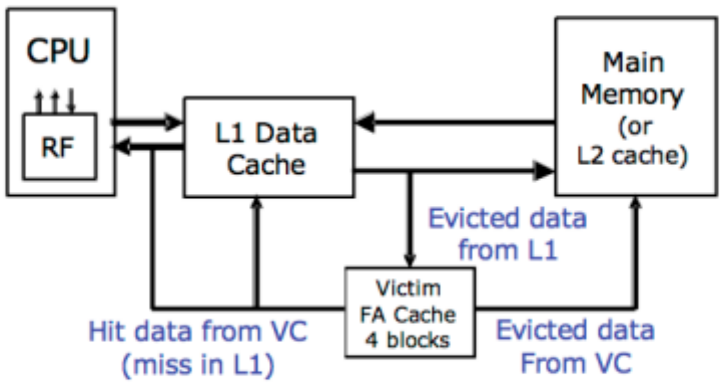
Microtags add an additional constraint to the cache: in a given cache set, all microtags must be unique. This constraint is necessary to avoid multiple microtag matches in the same set, which would prevent the cache from selecting the correct way.

State which of the 3C's of cache misses this constraint affects. **How will the cache miss rate compare to an ordinary 4-way set-associative cache? How will it compare to that of a direct-mapped cache of the same size?**

Problem 4: Victim Cache Evaluation

Although direct-mapped caches have an advantage of smaller access time than set-associative caches, they have more conflict misses due to their lack of associativity. In order to reduce these conflict misses, Norm Jouppi proposed victim caching, where a small fully-associative back up cache, called a victim cache, is added to a direct-mapped L1 cache to hold recently evicted cache lines.

The following diagram shows how a victim cache can be added to a direct-mapped L1 data cache. Upon a data access, the following chain of events takes place:



1. The L1 data cache is checked. If it holds the data requested, the data is returned.
2. If the data is not in the L1 cache, the victim cache is checked. If it holds the data requested, the data is moved into the L1 cache and sent back to the processor. The data evicted from the L1 cache is put in the victim cache, and put at the end of the FIFO replacement queue.
3. If neither of the caches holds the data, it is retrieved from memory, and put in the L1 cache. If the L1 cache needs to evict old data to make space for the new data, the old data is put in the victim cache and placed at the end of the FIFO replacement queue. Any data that needs to be evicted from the victim cache to make space is written back to memory or discarded, if unmodified.

Note that the two caches are *exclusive*. That means that the same data cannot be stored in both L1 and victim caches at the same time.

Problem 4.A — Baseline Cache Design

The diagram below shows our victim cache, a 32-Byte fully associative cache with four 8-Byte cache lines. Each line contains of two 4-Byte words and has an associated tag and two status bits (valid and dirty). The Input Address is 32-bits and the two least significant bits are assumed to be zero. The output of the cache is a 32-bit word.

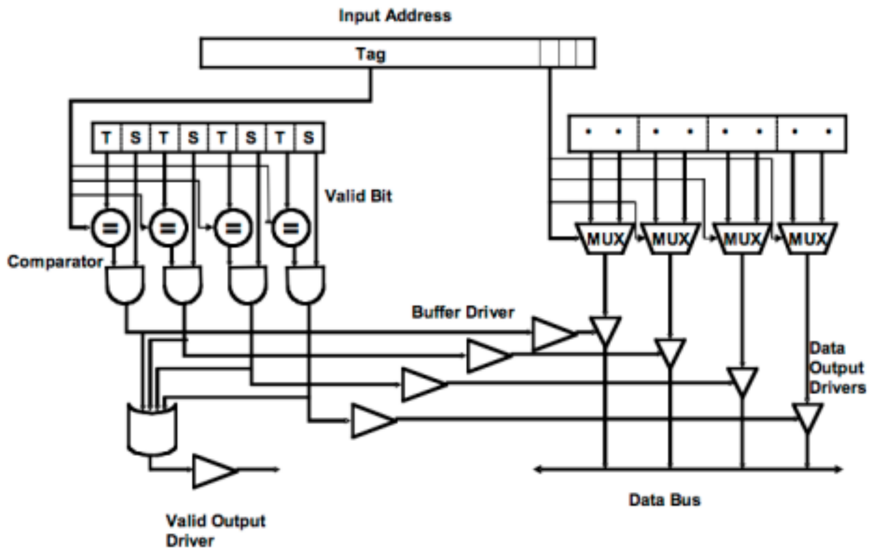


Figure 2.5-1: Victim cache datapath

Please complete Table 2.5-1 with delays across each element of the cache. Using the data you compute in Table 2.5-1, calculate the critical path delay through this cache (from when the Input Address is set to when both Valid Output Driver and the appropriate Data Output Driver are outputting valid data).

Component	Delay equation (ps)	FA(ps)
Comparator	20×(# of tag bits) + 100	
N-to-1 MUX	$50 \times \log_2 N + 100$	
Buffer driver	200	
AND gate	100	
OR gate	$50 \times \log_2 N + 100$	
Data output driver	50×(associativity) + 100	
Valid output driver	100	

Table 2.5-1: Delay of each cache component

Critical Path Cache Delay:

Problem 4.B — Victim Cache Behavior

Now we will study the impact of a victim cache on cache hit rate. Our main L1 cache is a 128 byte, direct-mapped cache with 16 bytes per cache line. The cache is word (4-bytes) addressable. The victim cache in Figure 2.5-1 is a 32-byte fully associative cache with 16 bytes per cache line, and is also word addressable. The victim cache uses the first in first out (FIFO) replacement policy.

Please complete Table 2.5-2 showing a trace of memory accesses. In the table, each entry contains the tag of that line, or “inv”, if no data is present. You should only fill in elements in the table when a value changes. For simplicity, the addresses are only 8 bits. The first 3 lines of the table have been filled in for you. For your convenience, the address breakdown for access to the main cache is depicted below.

7	6		4	3	2	1	0
TAG	INDEX			WORD SELECT		BYTE SELECT	

Input Address	Main Cache (tag)									Victim Cache (tag)		
	L0	L1	L2	L3	L4	L5	L6	L7	Hit?	Way0	Way1	Hit?
	inv	inv	inv	inv	inv	inv	inv	inv	-	inv	inv	-
0	0								N			N
80	1								N	0		N
4	0								N	8		Y
A0												
10												
C0												
18												
20												
8C												
28												
AC												
38												
C4												
3C												
48												
0C												
24												

Table 2.5-2: Memory access trace

Problem 4.C Average Memory Access Time

Assume **15%** of memory accesses are resolved in the victim cache. If retrieving data from the victim cache takes **5 cycles** and retrieving data from main memory takes **55 cycles**, by how many cycles does the victim cache improve the average memory access time?

Problem 5: Three C's of Cache Misses

Mark whether the following modifications will cause each of the categories to **increase, decrease**, or whether the modification will have **no effect**. You can assume the baseline cache is set associative. **Explain your reasoning**.

	Compulsory Misses	Conflict Misses	Capacity Misses
Double the associativity (capacity and line size constant)			
Halving the line size (associativity and # sets constant)			
Doubling the number of sets (capacity and line size constant)			

	Compulsory Misses	Conflict Misses	Capacity Misses
Adding prefetching			

Problem 6: Memory Hierarchy Performance

Mark whether the following modifications will cause each of the categories to **increase, decrease**, or whether the modification will have **no effect**. You can assume the baseline cache is set associative. **Explain your reasoning**.

	Hit Time	Miss Rate	Miss Penalty
Double the associativity (capacity and line size constant)			

Halving the line size (associativity and # sets constant)			
Doubling the number of sets (capacity and line size constant)			
Adding prefetching			