CS152 Computer Architecture and Engineering

Caches and the Memory Hierarchy

Assigned *02/08/2024*	Problem Set #2, Version (1.0)	*Due February 21 @ 11:59:59PT*

http://inst.eecs.berkeley.edu/~cs152/sp24

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 their own solution to the problems.

The problem sets also provide essential background material for the exam and the midterms. The problem sets will be graded primarily on an effort basis, but if you do not work through the problem sets yourself you are unlikely to succeed on the exam or midterms!

By grading primarily on an effort basis, we mean that we will award significant partial credit for demonstrating your understanding of the problem and concepts at hand. As long as reasonable assumptions and explanations are provided, we will lean towards awarding credit.

We will distribute solutions to the problem set after the deadline to give you feedback.

Assignments must be submitted through Gradescope by **11:59:59pm PT** on the specified due date. *Box/clearly mark all solutions that don't involve filling in a figure/table. Only boxed/clearly marked solutions and filled in figures/tables will be considered for grading.* See the course website for the policy on slip days (late submissions).

Name: _____

SID: _____

Collaborators (Name, SID):

_____

Problem 1: Cache Access-Time & Performance

This problem requires the knowledge of Handout #2 and the Lectures on Memory. Please, read these materials before answering the following questions.

Jessie is trying to determine the best cache configuration for a new processor. She 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. She 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

First, 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 256-KB (kibibyte = 2^{10} bytes) 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**. Use the ceiling of the logarithm to get an integer, if needed. *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	30×(# of index bits) + 80	Tag		
		Data		
Memory array	30× $\log_2$(# of rows) + 30× $\lceil \log_2 (\text{\# of bits in a row}) \rceil$ + 100	Tag		
		Data		
Comparator	30×(# of tag bits) + 70			
N-to-1 MUX	50×$\log_2$ N + 100			
Buffer driver	180			
Data output driver	50×(associativity) + 100			
Valid output driver	40			

Table 2.1-1: Delay of each Cache Component

i) **What is the critical path of this direct-mapped cache for a cache read?**
ii) **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.**
iii) **If the CPU clock is 2.5 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 256-KB (each way is 64KB), a 2-input gate delay is 50 ps, 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.**

i) **What is the critical path of the 4-way set-associative cache?**
ii) **What is the access time of the cache (the delay of the critical path)?**
iii) **What is the main reason that the 4-way set-associative cache is slower than the direct-mapped cache?**
iv) **If the CPU clock is 2.5 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 32 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 by filling in the hexadecimal tag values** 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.*

Address in Binary

D-map	**Addresses and tags are in HEX**								
	line in cache (tag)								hit?
Address	L0	L1	L2	L3	L4	L5	L6	L7	
11B	1	inv	inv	inv	inv	inv	inv	inv	no
134		1							no
20D	2								no
1A2									
105									
360									
27D									
121									
1A3									
17A									
307									
273									
131									

	Direct-Mapped
Total Misses	
Total Accesses	

Address in Binary	**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**	
	11B	4	inv	inv	inv	inv	inv	inv	inv	no
	134					4				no
	20D		8							no
	1A2									
	105									
	360									
	27D									
	121									
	1A3									
	17A									
	307									
	273									
	131									

	4-way LRU
Total Misses	
Total Accesses	

Address in Binary	**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**	
	11B	4	inv	inv	inv	inv	inv	inv	inv	no
	134					4				no
	20D		8							no
	1A2									
	105									
	360									
	27D									
	121									
	1A3									
	17A									
	307									
	273									
	131									

	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 256-KB caches studied in 1.A and 1.B.

i) What would be the average memory access latency in CPU cycles for each cache? Assume that the cache miss penalty is 20 cycles and use cache access cycle count from 1.A and 1.B. Which one is better?
ii) 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.
iii) 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:

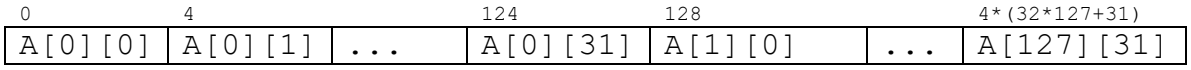
Loop A	***Loop B***

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



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 an 8KB direct-mapped data cache with 4-word (16-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 4-word (16-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 8KB set-associative data cache with 4 ways, and 4-word (16-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 Handout #2. Figure 1, below, shows the modified tag comparison and driver hardware in the microtagged cache.

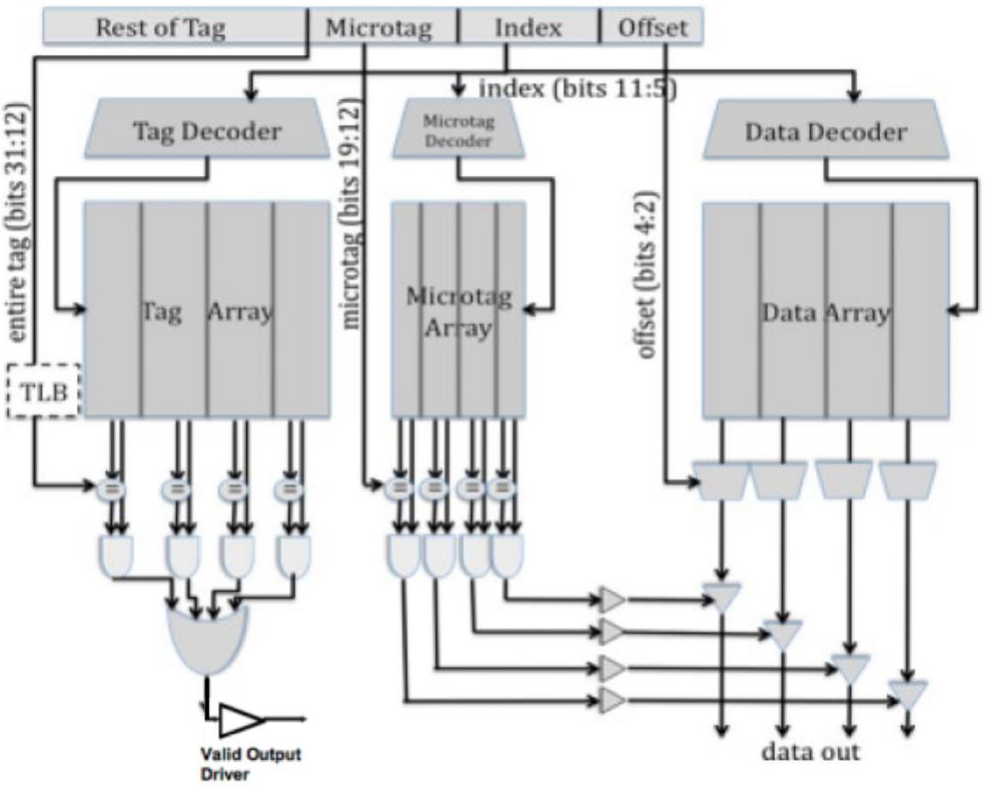


Figure 2.4-1: Microtagged cache datapath

Problem 3.A **Cache Cycle Time**
(PRACTICE - OPTIONAL)

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).

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	240
		Data	240	240
		Microtag		240
Memory array	$20\times\log_2$ (# of rows) + $20\times\log_2 \lceil$(# of bits in a row)$\rceil$ + 100	Tag	380	380
		Data	440	440
		Microtag		340
Comparator	20×(# of tag bits) + 100	Tag	500	500
		Microtag		260
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

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

ii) 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, 90%, and the same average miss penalty, 15 ns. Using the cycle times 1.5 ns and 1.2 ns for the baseline and microtag caches respectively, compute the average memory access time for both caches.

i) What was the old baseline AMAT (in ns)?

ii) What is the new AMAT (in ns)?

Problem 3.C **Constraints**

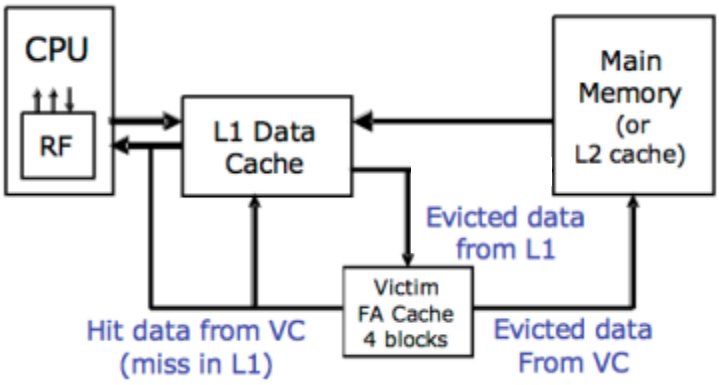
Microtags add an additional constraint to the cache: in a given cache set, all microtags must be unique. This constraint is necessary to avoid multiple microtag matches in the same set, which would prevent the cache from selecting the correct way.

i) State which of the 3C's of cache misses this constraint affects.
ii) How will the cache miss rate compare to an ordinary 4-way set-associative cache?
iii) How will it compare to that of a direct-mapped cache of the same size?
iv) Which 8 bits of the tag might you want to use for the microtag and why?

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**
(PRACTICE - OPTIONAL)

The diagram below shows our victim cache, a 32-byte fully associative cache with four 8-byte cache lines. Each line contains two 4-byte words and has an associated tag and two status bits (valid and dirty). The Input Address is 32-bits. Since the cache is word-addressed, it does not use the two least significant bits. The output of the cache is a 4-byte word.

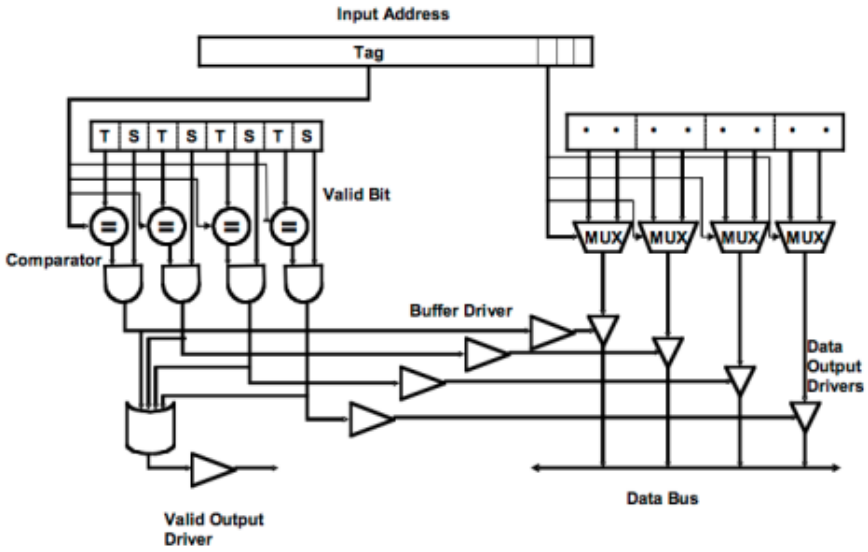


Figure 2.5-1: Victim cache datapath

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	30×(# 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	

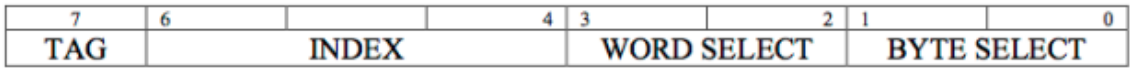
Table 2.5-1: Delay of each cache component

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 is similar to the one in Figure 2.5-1. It is a 32-byte fully associative cache with 16 bytes per cache line and is also word addressable. (Note that these parameters are different from 4.A.) It 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.



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 L1 misses are resolved in the victim cache. If retrieving data from the victim cache takes **4 cycles** and retrieving data from main memory takes **50 cycles**, by how many cycles does the victim cache improve the average memory access time? Assume that the L1 miss rate is **10%**.

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**.
For subparts where the outcome is ambiguous, pick one outcome and answer with reasonable assumptions and explanations.

	Compulsory Misses	Conflict Misses	Capacity Misses
Halving the line size (associativity and # sets constant)			
Doubling the number of sets (capacity and line size constant)			
Adding good prefetching			
Combine ICache and DCache into a single L1 cache with the combined capacity (associativity and line size constant)			

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**.
For subparts where the outcome is ambiguous, pick one outcome and answer with reasonable assumptions and explanations.

	Hit Time	Miss Rate	Miss Penalty
Halving the line size (associativity and # sets constant)			
Doubling the number of sets (capacity and line size constant)			
Adding good prefetching			
Combine L1ICache and L1DCache into a single L1 cache with the combined capacity (associativity and line size constant)			