ECE 4750 Computer Architecture

Topic 4: Fundamental Memory Microarchitecture

http://www.csl.cornell.edu/courses/ece4750
School of Electrical and Computer Engineering
Cornell University

revision: 2022-10-16-22-23

List of Problems

Problem 1. Impact of Cache Access Time and Replacement Policy

In this problem, we will be comparing various microarchitectures for a simple data cache. For all parts, the memory requests use a 32-bit byte address, although you should assume that all addresses are word aligned. Since words are four bytes, this means the bottom two bits of the address will always be zero. All caches contain exactly eight cache lines, and each cache line contains four words (i.e., each cache line is 16 bytes long). Thus the total cache capacity is $8 \times 16\,\text{B} = 128\,\text{B}$.

Some parts of this problem require you to identify the critical path of a specific microarchitecture. Figure 1 lists simplified delay equations for the cache hardware components. These delay equations are parameterized by the size of each component. Delay is measured in normalized gate delays, where $1\,\tau$ is the delay of a single inverter driving four identical inverters. To simplify things, assume that the delay of a component is always the same regardless of the order in which different inputs arrive at a component. More specifically, the delay of a write access is the same regardless of whether the address or write enable arrives before the write data or vice versa. Also assume that we are using combinational memories (i.e., the address is set and the data is returned on the same cycle). Note that the $\lceil x \rceil$ notation denotes the ceiling operator, i.e., the value x rounded up to the next largest integer.

As discussed in lecture, you should always assume a single read/write port for the data memories (i.e., we can only do a single access per cycle, it is not possible to both read and write the data memory in the same cycle). The data memories in all cache microarchitectures have the following ports:

- line select : one-hot encoding of which line to read/write
- write data : 16 B of write data
- read data : 16 B of read data
- write enable : one bit indicating if we should actually write the write data
- word write enable : two-bit encoding of which word to write

Component	**Delay (τ)**	**Comment**
register read	1	delay from clock edge to data out
register write	1	setup time constraint
n-input AND gate	$n-1$	simple logic gate
n-input OR gate	$n-1$	simple logic gate
n-to-m decoder	$3+2n$	3 for fixed overhead, $2n$ for logic
n-bit comparator	$3+2\lceil \log\_2(n) \rceil$	3 for initial XOR gate, $2\lceil \log\_2(n) \rceil$ for OR tree
n-input m-bit mux	$3\lceil \log\_2(n) \rceil + \lceil m/8 \rceil$	$3\lceil \log\_2(n) \rceil$ for tree-based muxing logic, $\lceil m/8 \rceil$ for interconnect
$n \times m$ memory access	$10 + \lceil (n+m)/16 \rceil$	n rows and m bits per row, 10 for fixed overhead and bitcell access, $\lceil (n+m)/16 \rceil$ to drive word and bit lines

Figure 1: Simplified and Parameterized Delay Equations for Cache Components

Part 1.A Miss Rate Analysis

We start by studying the miss rate for a direct-mapped and two-way set-associative caches when processing the same sequence of memory requests. We will consider both LRU and FIFO replacement policies for the set-associative cache. The miss rate is independent of how we order the tag check with respect to the data access, so this part can apply to either of the cache microarchitectures discussed in the next parts. Remember, both caches have exactly eight cache lines, and each cache line contains four words. All caches begin with every line invalidated.

We will be using the following sequence of 20 memory requests. The addresses are specified as hexadecimal byte addresses. Assume that each request is to read a single aligned word of data, so the least significant two bits are always zero.

```
0x024, 0x030, 0x07c, 0x070, 0x100, 0x110, 0x204, 0x214, 0x308, 0x110,
0x114, 0x118, 0x11c, 0x410, 0x110, 0x510, 0x110, 0x610, 0x110, 0x710
```

To illustrate how the cache contents change over time, we will be using a table similar to the ones in Figures 2–4 with one column for each tag in the tag array and one row for each memory request. The contents of each cell in the table should indicate the *tag of the cache line* currently located at that location in the cache *before the transaction has executed*. Assume the set index bits are positioned in the address as shown in Figures 6 and 7. Use a dash (–) to indicate an invalid cache line. *You only need to fill in elements in the table when the value changes!* To get you started, we have filled in the three tables in Figures 2–4 correctly for the first three memory requests. Study these first few memory requests to understand how to fill out the rest of the table. Each table should have a total of 21 rows, one for each memory request and one at the end in case you need to show a final state update. *Remember that the tag entries should always reflect the state of the tag array before the corresponding transaction on that row executes!* After filling in the table, also enter the number of misses and the miss rate at the bottom of the table.

Transaction Address	tag	idx	m/h	L0	L1	L2	L3	L4	L5	L6	L7
0x024	0x00	0x2	m	-	-	-	-	-	-	-	-
0x030	0x00	0x3	m			0x00					
0x07c	0x00	0x7	m				0x00				
0x070	0x00	0x7	h								0x00
0x100	...										
Number of Misses =											
Miss Rate =											

Figure 2: Direct-Mapped Cache Contents Over Time

Transaction				Set 0			Set 1			Set 2			Set 3		
Address	tag	idx	m/h	U	Way 0	Way 1	U	Way 0	Way 1	U	Way 0	Way 1	U	Way 0	Way 1
0x024	0x00	0x2	m	-	-	-	-	-	-	-	-	-	-	-	-
0x030	0x00	0x3	m							0	0x00				
0x07c	0x01	0x3	m										0	0x00	
0x070	0x01	0x3	h										1		0x01
0x100	...				...										
Number of Misses =															
Miss Rate =															

Figure 3: Two-Way Set-Associative Cache Contents Over Time with LRU Replacement

Transaction				Set 0		Set 1		Set 2		Set 3	
Address	tag	idx	m/h	Way 0	Way 1	Way 0	Way 1	Way 0	Way 1	Way 0	Way 1
0x024	0x00	0x2	m	-	-	-	-	-	-	-	-
0x030	0x00	0x3	m					0x00			
0x07c	0x01	0x3	m							0x00	
0x070	0x01	0x3	h								0x01
0x100	...			...							
Number of Misses =											
Miss Rate =											

Figure 4: Two-Way Set-Associative Cache Contents Over Time with FIFO Replacement

Part 1.B Sequential Tag Check then Memory Access

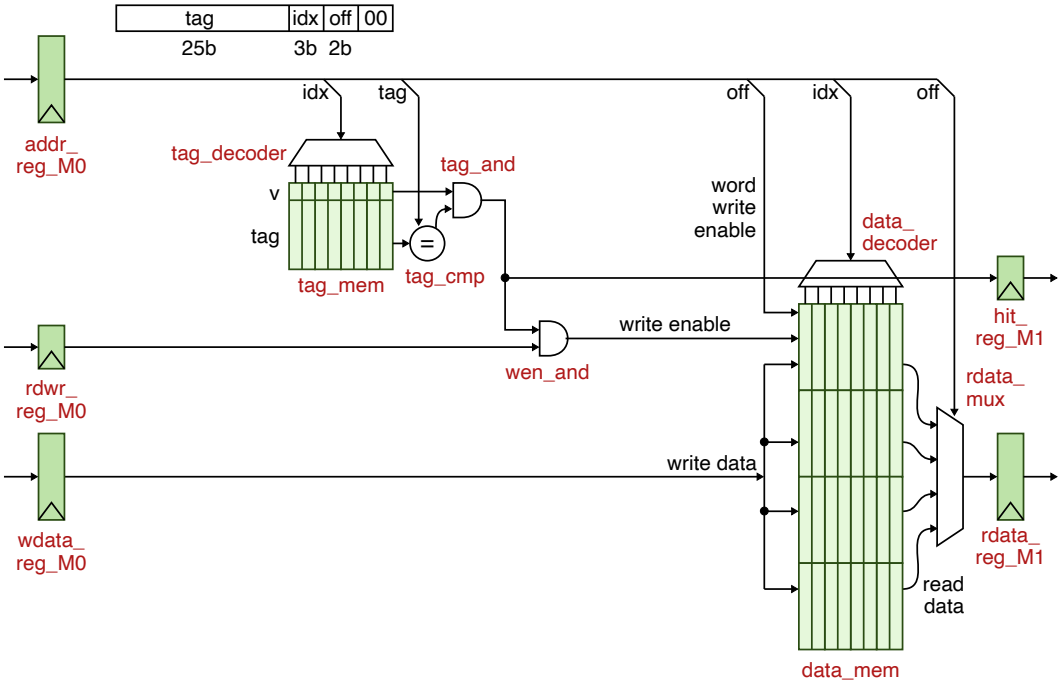
Figure 6 illustrates two cache microarchitectures that serialize the tag check before data access. This means that for both reads and writes, the cache completely finishes the tag check and accesses the data memory only on a cache hit. Figure 6(a) is for a directed-mapped cache, while Figure 6(b) is for a two-way set-associative cache.

We now want to determine the critical path and cycle time in units of τ for each cache microarchitecture. As an example, Figure 5 shows the critical path and cycle time for the directed-mapped cache in Figure 6(a). Spend some time understanding the microarchitectural diagrams in Figure 6 and the example read/write access time table in Figure 5 before continuing to work on this problem. Note that the tag is 25 bits, but each row of the tag memory requires 26 bits since it must also include a valid bit. Also note that even for reads we have to wait for the delay though the `wen_and` gate because we are assuming that we must wait for the last input to the data memory to stabilize before the delay of the data memory begins.

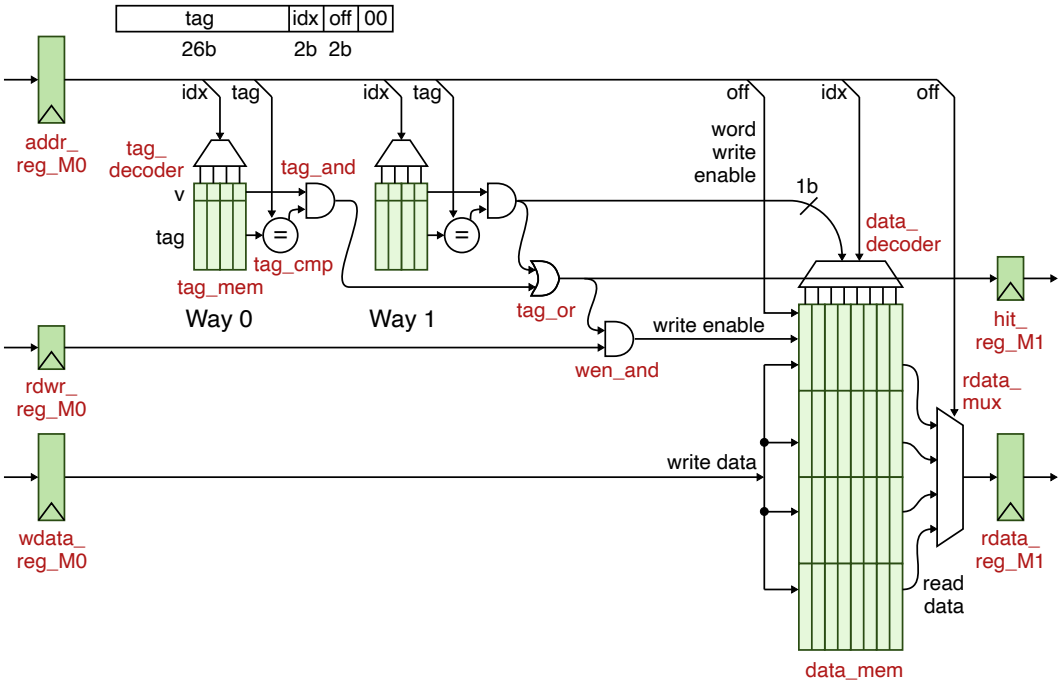
Create a table similar to the one shown in Figure 5 which identifies the critical path and cycle time in units of τ for the two-way set-associative cache in Figure 6(b). Compare the cycle times of the two cache microarchitectures. What is the primary reason one microarchitecture is slower than the other microarchitecture?

Component	Delay Equation	Delay (τ)
`addr_reg_M0`	1	1
`tag_decoder`	$3 + 2 \cdot 3$	9
`tag_mem`	$10 + \lceil (8+26)/16 \rceil$	13
`tag_cmp`	$3 + 2\lceil \log\_2(25) \rceil$	13
`tag_and`	1	1
`wen_and`	1	1
`data_mem`	$10 + \lceil (8+128)/16 \rceil$	19
`rdata_mux`	$3\lceil \log\_2(4) \rceil + \lceil 32/8 \rceil$	10
`rdata_reg_M1`	1	1
Total		**68**
`addr_reg_M0`	1	1
`tag_decoder`	$3 + 2 \cdot 3$	9
`tag_mem`	$10 + \lceil (8+26)/16 \rceil$	13
`tag_cmp`	$3 + 2\lceil \log\_2(25) \rceil$	13
`tag_and`	1	1
`wen_and`	1	1
`data_mem`	$10 + \lceil (8+128)/16 \rceil$	19
Total		**57**

Figure 5: Critical Path and Cycle Time for Direct-Mapped Cache with Serialized Tag Check before Data Access – Corresponds to cache microarchitecture in Figure 6(a), top path is for reads, while bottom path is for writes. Reads are the critical path.



(a) Direct-Mapped Cache



(b) Two-Way Set-Associative Cache

Figure 6: Cache Microarchitectures for Sequential Tag Check then Data Access

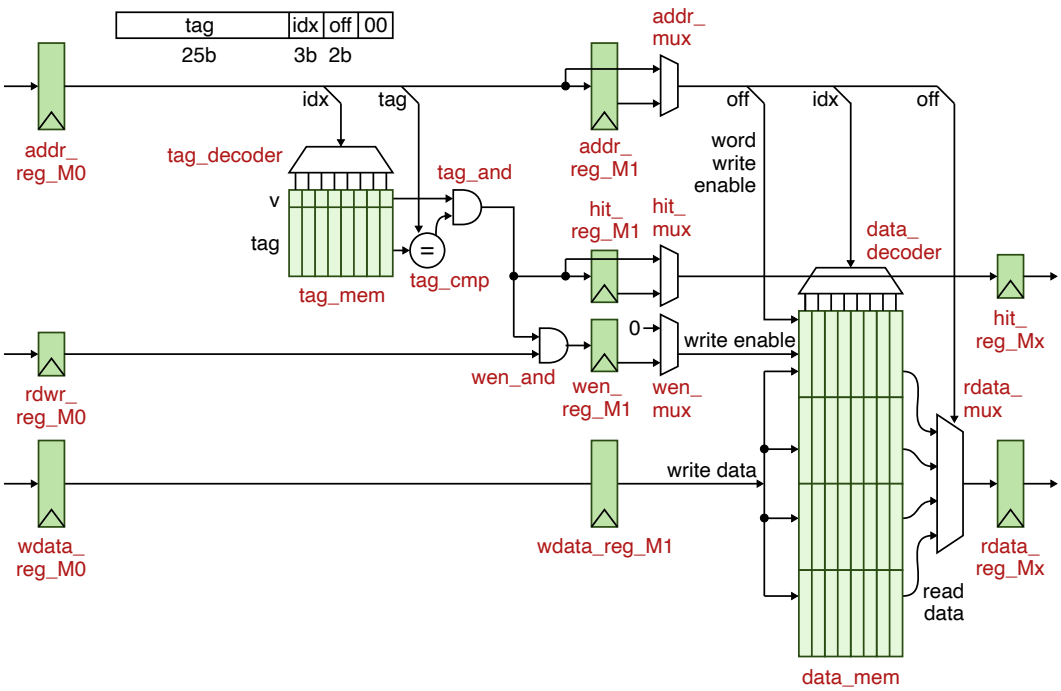
Part 1.C Parallel Read Hit Path

Figure 7 illustrates two cache microarchitures with parallel read hit paths and pipelined write hit paths. This means that for a single read request, the tag check is done in parallel with the data memory read access, while for a single write request the tag check is done in stage M0 and the data memory write access is done in stage M1. Figure 7(a) is for a directed-mapped cache, while Figure 7(b) is for a two-way set-associative cache. Spend some time understanding this microarchitectural diagram before continuing to work on this problem.

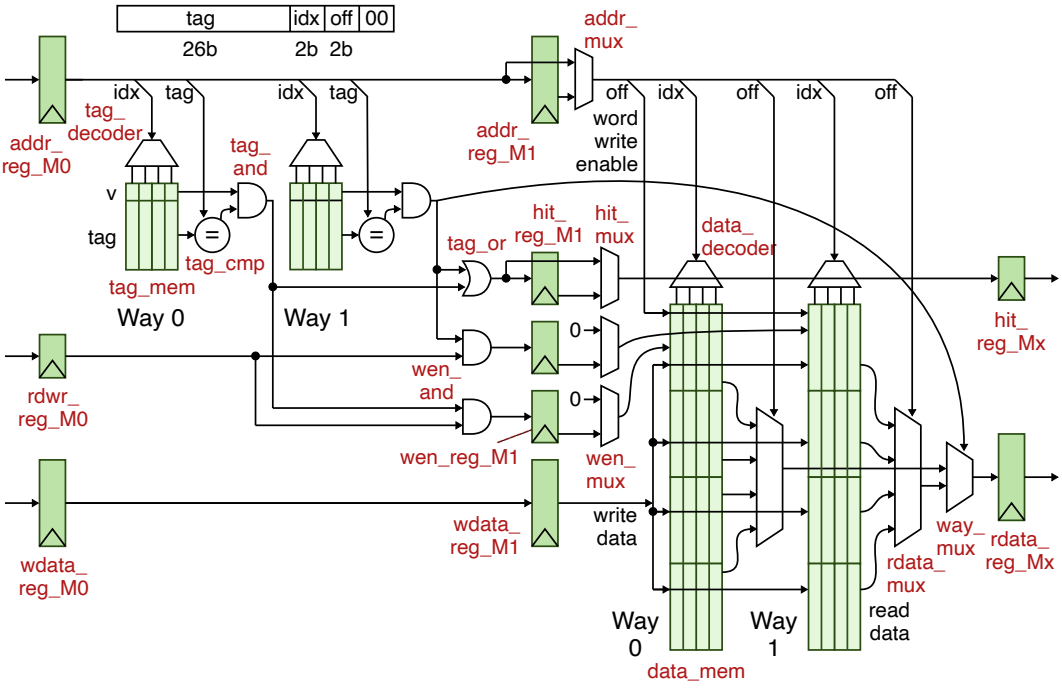
For this part we will *focus just on the parallel read hit path* for both the direct-mapped and set-associative caches. Create two tables similar to the one shown in Figure 5 which identifies the critical path and cycle time in units of τ for just the parallel read hit paths. Note that since the tag check and the data memory read access are done in parallel, you will need to examine both of these paths to determine which one is in fact the critical path. Compare the cycle times of the two cache microarchitectures. **What is the primary reason one microarchitecture is slower than the other microarchitecture?**

Part 1.D Pipelined Write Hit Path

For this part we will *focus just on the pipelined write hit path* for both the direct-mapped and set-associative caches shown in Figure 7. Create two tables similar to the one shown in Figure 5 which identifies the critical path and cycle time in units of τ for just the pipelined write hit path. Note that since the tag check and the data memory write access happen in two different stages, you will need to examine both of these paths to determine which one is in fact the critical path. Compare the cycle times of the two cache microarchitectures. **What is the primary reason one microarchitecture is slower than the other microarchitecture?**



(a) Direct-Mapped Cache



(b) Two-Way Set-Associative Cache

Figure 7: Cache Microarchitectures for Parallel Reads and Pipelined Writes

Part 1.E Average Memory Access Latency

Cache performance is impacted by many factors. In this part, we put together the access time analysis from Parts 1.B-1.D and the miss rate analysis from Part 1.A to determine the average memory access latency for six different cache configurations. Our goal is to determine which configuration achieves the lowest average memory access latency for the sequence of memory requests given in Part 1.A. Remember that the average memory access latency is defined as:

$$\text{Avg Mem Access Latency} = \text{Hit Time} + (\text{Miss Rate} \times \text{Miss Penalty})$$

For this problem you can assume the miss penalty is 300τ. **Create a table similar to the one in Figure 8. Fill in the hit time in τ.** Use Parts 1.B-1.D, miss rate (use Part 1.A), and the average memory access time in τ for each of the six configurations. The sequence of memory requests under consideration consists only of read requests, so the hit time is always one cycle. Assume the cache hit path is the critical path for the entire processor, and thus sets the cycle time. Since the replacement policy impacts the miss path, you can assume it has no impact on the critical path. Note that the critical path will be different for each microarchitecture. For a microarchitecture that serializes the tag check before data access, the critical path was determined in Part 1.B. For a microarchitecture with a parallel read hit path and a pipelined write hit path, the critical path will either be through the read hit path (Part 1.C) or the write hit path (Part 1.D), whichever is longest. Remember that for the write hit path, you need to consider the longer of the two pipeline stages. **Which configuration has the lowest average memory access time? How general is your conclusion? Can we safely say that we should always choose this configuration regardless of the application requirements and/or technology constraints?**

Associativity	µarch	Replacement Policy	Hit Time (τ)	Miss Rate (ratio)	Miss Penalty (τ)	AMAL (τ)
Direct Mapped	Seq	n/a	68		300	
2-way Set Assoc	Seq	LRU			300	
2-way Set Assoc	Seq	FIFO			300	
Direct Mapped	PP	n/a			300	
2-way Set Assoc	PP	LRU			300	
2-way Set Assoc	PP	FIFO			300	

Figure 8: Average Memory Access Latency for Six Cache Configurations – Seq = serialize tag check before data access, PP = parallel read hit path and pipelined write hit path, AMAL = average memory access latency