Problem 1: Cache Access-Time & Performance

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 1. 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).*

现在我们要计算直接映射缓存的访问时间。 我们使用图 1 中所示的实现。假定具有 8 字（32 字节）高速缓存行的 128 KB 高速缓存。

一行 32 字节, 八个字,

128KB /32B = 4K 行.

Index log2(4K) = 12 bits

该地址为 32 位，并且是字节寻址的，因此该地址的两个最低有效位将被忽略，因为高速缓存访问是按字对齐的。 数据输出也是 32 位（1 个字），并且 MUX 在高速缓存行的 8 个字中选择一个字 N = 8 。 使用表 2.1-1 中给出的延迟方程，在表中填写直接映射（DM）高速缓存的列。 在数据输出驱动程序的方程式中，“关联性”是指高速缓存的关联性（对于直接映射的高速缓存，为 1；对于 A 向集合关联的高速缓存，为 A）。

Offset = 5bits, *index = log2（128KB/32B） = log2（4K）=12 bits, tag =32- index –offset = 15 bits, 加上两个状态位,* # of bits in a row = *17bits,*

4way 一个set 有4 路, 4K 行-> 1K sets, 4-way set-associative caches, *index = log2（128KB/32B/4） = log2（1K）=10 bits , tag =17bits,*

Tag # of bits in a row = *19*4= 76bits,*

Data # of bits in a row = 4* 32bytes 一行 = 1024bits

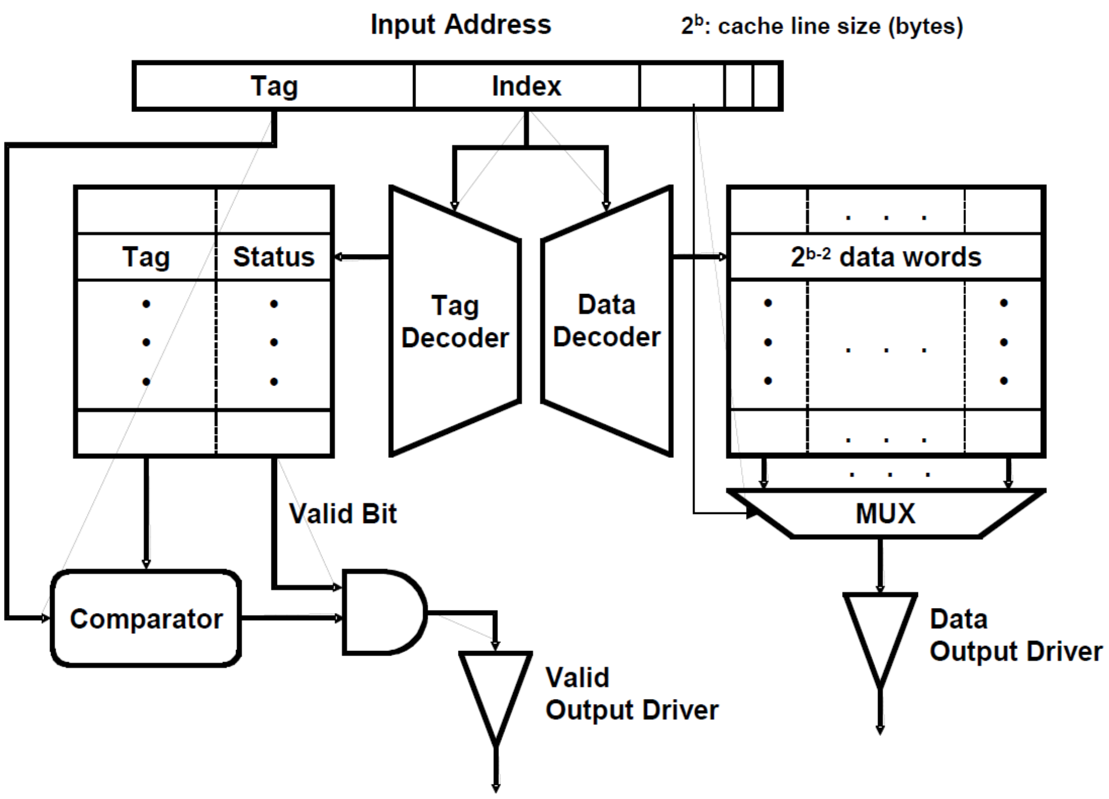


Figure 1 A direct-mapped cache implementation

Component	Delay equation (ps)		DM (ps)	SA (ps)
Decoder	20×(# of index bits) + 100	Tag	20*12+100=340	300
		Data	340	300
Memory array	20× $\log_2$ (# of rows) + 20× $\log_2$ (# of bits in a row) + 100	Tag	20*12+20*log2(17)+100= 422	20*10+20*log2(76)+100= 425
		Data	20*12+20*log2(32*8)+100 = 500	500
Comparator	20×(# of tag bits) + 100		20*15+100= 400	20*17+100= 400
N-to-1 MUX	50×$\log_2$ N + 100		250	50*log2(8)+100 = 250
Buffer driver	200			200
Data output driver	50×(associativity) + 100		150	300
Valid output driver	100		100	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?

解:

tag check valid driver

(tag decode time) + (tag memory access time) + (comparator time) + (AND gate time) + (valid output driver time)

= 340 + 422 + 400 + 50 + 100 = 1312 ps

data output driver

(data decode time) + (data memory access time) + (mux time) + (data output driver time) = 340 + 500 + 250 + 150 = 1240 ps

tag check is the critical path. (1312 ps / (1 / 1.5GHz)) = 2 cycles.

Problem 1.B Access Time: Set-Associative

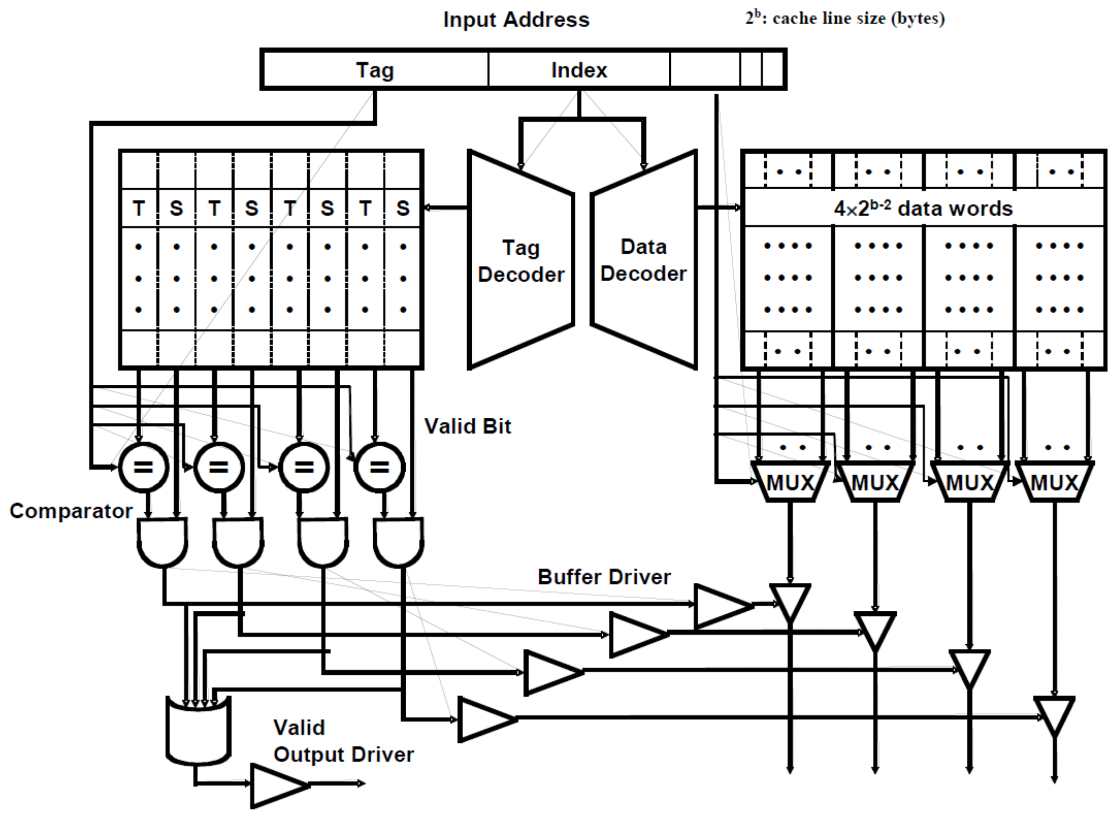


Figure 2 A 4-way set-associative cache implementation

We also want to investigate the access time of a set-associative cache using the 4-way set-associative cache in Figure 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.

我们还想使用图 2 中的 4 路 set-associative 缓存来研究 set-associative 缓存的访问时间。假设总缓存大小仍为 128 KB（每种方式为 32byes），即 4 输入 门延迟为 100 ps，所有其他参数（如输入地址，高速缓存行等）与 2.1.A 部分相同。计算每个组件的延迟，并在表 2.1-1 中填写 4 路组关联缓存的列。4 路组关联缓存的关键路径是什么？ 缓存的访问时间是多少（关键路径的延迟）？比直接映射高速缓存慢的主要原因是什么？ 如果 CPU 时钟为 1.5 GHz，则高速缓存访问需要花费多少个 CPU 周期?

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?

解:
valid output driver
(tag decode time) + (tag memory access time) + (comparator time) + (AND gate time) + (OR gate time) + (valid output driver time) = 300 + 425 + 440 + 50 + 100 + 100 = 1415 ps

data output driver (tag)
(tag decode time) + (tag memory access time) + (comparator time) + (AND gate time) + (buffer driver time) + (data output driver) = 300 + 425 + 440 + 50 + 200 + 300 = 1715 ps

data output driver (data)
(data decode time) + (data memory access time) + (mux time) + (data output driver) = 300 + 500 + 250 + 300 = 1350 ps
data output driver (tag) **is the critical path. (1715 ps / (1 / 1.5GHz)) = 3 cycles.**

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.* 现在，Ben 正在研究集关联性对缓存性能的影响。 由于他现在知道每种配置的访问时间，因此他想知道每种配置的未命中率。 对于未命中率分析，Ben 正在考虑使用两个小型缓存：具有 8 行（每行 16 字节）的直接映射缓存和具有相同大小和行大小的 4 路集关联缓存。 对于集合关联缓存，Ben 尝试了两种替换策略–最近最少使用（LRU）和循环（FIFO）。Ben 通过访问以下十六进制字节地址序列（从空缓存开始）来测试缓存。 为简单起见，假定地址仅为 12 位。 对于直接映射的高速缓存和两种类型的 4 路集关联高速缓存，请完成以下表格，以显示随着访问发生而高速缓存内容的进程（在表中，“ inv” =无效，并且特定高速缓存行的列包含 该行的标签）。 同样，为每个地址计算标签和索引（这将有助于填写表格）。 您只需在值更改时填写表中的元素。

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			3						No
102	2								No
361							6		No
204	4								No
114									Yes
1A4									Yes
177								2	No
301	6								No
206	4								No

135									yes

	D-map
Total Misses	10
Total Accesses	13

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		0D							
102			08						
361				1B					
204									Yes
114									Yes
1A4									Yes
177							0B		No
301			18						No
206									Yes
135									yes

	4-way LRU
Total Misses	8
Total Accesses	13

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		0D							No
102			08						No
361				1B					No
204									Yes
114									Yes
1A4									Yes
177							0B		No
301	18								No
206		10							No
135									yes

	4-way FIFO
Total Misses	9
Total Accesses	13

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.

假设以上分析的结果可以表示在 1.A 和 1.B 中研究的直接映射和 4 路集关联 128 KB 高速缓存的平均未命中率。 每个高速缓存在 CPU 周期中的平均内存访问延迟是多少（假设高速缓存未命中惩罚为 20 个周期）？ 哪一个更好？ 对于集合关联缓存的不同替换策略，对于 1.C 中的地址流，哪一个具有较小的缓存未命中率？ 解释为什么。 更换政策是否总是会产生更好的漏检率？ 如果不是，请使用地址流给出一个计数器示例。

解:

The miss rate for the direct-mapped cache is 10/13.

The miss rate for the 4-way LRU set associative cache is 8/13.

AMAT = (hit time) + (miss rate) *(miss penalty).

direct-mapped cache AMAT = (2 cycles) + (10/13) × (20 cycles) = 17.4 cycles.

LRU (3 cycles) + (8/13) × (20 cycles) = 15.3 cycles.

FIFO (3 cycles) + (9/13) × (20 cycles) = 16.8 cycles.

The set-associative cache with LRU replacement is better.

LRU does not always going to yield better miss rates.

Problem 2: Loop Ordering

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: 128 行 32 列

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(连续地) 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: 一个 integer 4bytes32 位.

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.

Cache has 4K/32 = 32 lines ,32 sets, index 5 bits, offset log2(32) = 5 bits.

A 一开始访问[0][0],miss, 把 32bytes 给 load 进 set0 来, 之后 [0][1],[0][2] [0][3] – [0][7]都不会 miss. 然后[0][8] miss,放在 set1, … [0][16] miss,放在 set2, … 一行 miss4 次.一行 4 个 set, 8 行填满了 cache.

第九行, [8][0]放在 set0, [8][4]放在 set1, 每行 4 次 miss, 4*128=512

B 一开始访问[0][0],miss, 把 32bytes 给 load 进来 set0, 之后 [1][0] miss，因为 128 对应是 set4,把 32bytes 给 load 进 set4, [2][0] 又 miss,放 set8..[8][0] miss, 替换 set0.事实上每次访问都是 miss,而且 evict 也非常多.

The number of cache misses for Loop A: ______**512**_______________________

The number of cache misses for Loop B: ____________**4096**_________________

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.

A 128 行填满了 cache.

第九行, [8][0]放在 set32, [8][4]放在 set33, 如果全都可以放下就不会有 conflict miss,

B 一开始访问[0][0],miss, 把 32bytes 给 load 进来 set0, 之后 [1][0] miss，因为 128 对应是 set4,把 32bytes 给 load 进 set4, [2][0] 又 miss,放 set8..[8][0] miss, 放在 set32 要让 128 行都放的下, 要 4*128=512 行

Data-cache size required for Loop A: _________512___________________ cache line(s)

Data-cache size required for Loop B: ___________512_________________ 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.

FIFO 是先入先出.

Cache has 4K/32 = 32 lines ,32 sets, index 5 bits, offset log2(32) = 5 bits.

A 一开始访问[0][0],miss, 把 32bytes 给 load 进 set0 来, 之后 [0][1],[0][2] [0][3] –[0][7]都不会 miss. 然后[0][8] miss,放在 set1, … [0][16] miss,放在 set2, … 一行 miss4 次.一行 4 个 set, 8 行填满了 cache.

第九行, [8][0]放在 set0, [8][4]放在 set1, 每行 4 次 miss, 4*128=512

B 一开始访问[0][0],miss, 把 32bytes 给 load 进来 set0, 之后 [1][0] miss , 全相连可以选择 set1,把 32bytes 给 load 进 set1, [2][0] 又 miss,放 set2..

[32][0] miss, 替换 set0.

[64][0]miss, [96][0] miss,事实上每次访问依旧都是 miss,而且 evict 也非常多.

The number of cache misses for Loop A:_____________512________________

The number of cache misses for Loop B:____________4096______________

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. 在这个问题中，我们探索了微标记，一种减少集合关联缓存访问时间的技术。 回想一下，对于关联的高速缓存，必须在将加载结果返回给 CPU 之前完成标签检查，因为标签检查的结果决定了选择哪种高速缓存方式。 因此，标签检查通常位于关键路径上。

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. 进行标签检查（选择方式）的时间在很大程度上取决于标签的大小。 我们可以通过仅检查标签的一个子集（称为微标签）并使用比较结果来选择适当的缓存方式，从而加快方法选择的速度。 当然，还必

须进行完整标签检查，以确定高速缓存访问是命中还是未命中，但是此比较与选择方式并行进行。 我们将完整标签与微标签阵列分开存储。

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 2.   Figure 3, below, shows the modified tag comparison and driver hardware in the microtagged cache. 我们将考虑微标记对具有 32 字节行的 4 路集关联 16KB 数据高速缓存的影响。 地址为 32 位长。 Microtag 的长度为 8 位。 基线缓存（即无微标签）如图 2 所示。下面的图 3 显示了微标签缓存中的修改后的标签比较和驱动程序硬件。

解:

一行 32 字节, 16KB/32B = 32*16 = 512 行, , 4way,所以是 128 sets, index 是 log2(128) = 7 bits, offset = log2 (32) = 5 也就是[4:0]   ,tag = 20bits, microtag = 8bits, rest of tag =12bits.

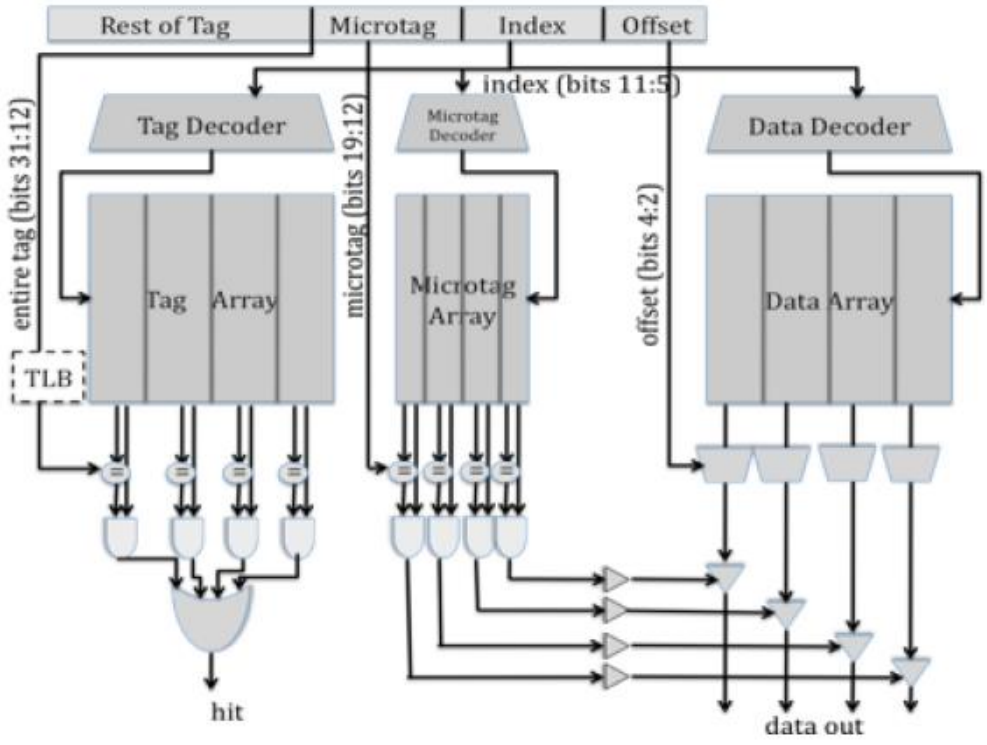


Figure 3 Microtagged Cache

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.** 下表 2.4-1 包含了 4 路集关联高速缓存中基线和微标签高速缓存中组件的延迟。 对于这两种配置，请确定关键路径和缓存访问时间（即，通过关键路径的延迟）。 填写微标签缓存的列。

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(20*7+100)	240(20*7+100)
		Data	240	240(20*7+100)
Memory array	$20\times\log_2$ (# of rows) + $20\times\log_2$ (# of bits in a row) + 100	Tag	369	180+20*log2(20)+100 = 180+86.4+100 =369
		Data	440	440
		Microtag	▨	180+20*log2(8)+100= 180+60+100 =340
Comparator	20×(# of tag bits) + 100	Tag	500	500
		Microtag	▨	20*(8)+100 =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

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

解:

Full tag check tag 240 ps + 369 ps + 500 ps + 50 ps + 100 ps + 100 ps = 1359 ps

data select based on the full tag match: 240 ps + 369 ps + 500 ps + 50 ps + 200 ps + 300 ps = 1659 ps

Data readout data decoder = 240 ps + 440 ps + 250 ps + 300 ps = 1230 ps

The critical path is the data select based on the full tag match. The cycle time is 1659 ps.

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

Full tag check same as baseline => 1359 ps

Data select based on microtag check = 240 ps + 340 ps + 260 ps + 50 ps + 200 ps + 300 ps = 1390 ps

Data readout same as baseline => 1230 ps

The critical path is the data select based on the microtag. The cycle time is 1390 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)?** 暂时假定基线缓存和微标记缓存具有相同的命中率（95%）和相同的平均未命中惩罚（20 ns）。 使用 3.A 中计算的周期时间作为命中时间，计算两个缓存的平均内存访问时间。

AMAT = (hit_time) + (miss)_rate * (miss_penalty)
Old AMAT = 1.62 + 0.05*20ns = 2.62 ns
New AMAT with microtags = 1.356 + 1 = 2.356 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?**
微标签为缓存增加了一个额外的约束：在给定的缓存集中，所有微标签都必须是唯一的。 为了避免同一组中的多个微标签匹配，此约束是必需的.指出哪个 3C 缓存错过了此限制影响的范围。 高速缓存未命中率与普通的 4 路集关联高速缓存相比如何？ 与具有相同大小的直接映射缓存相比，它将如何?

解:

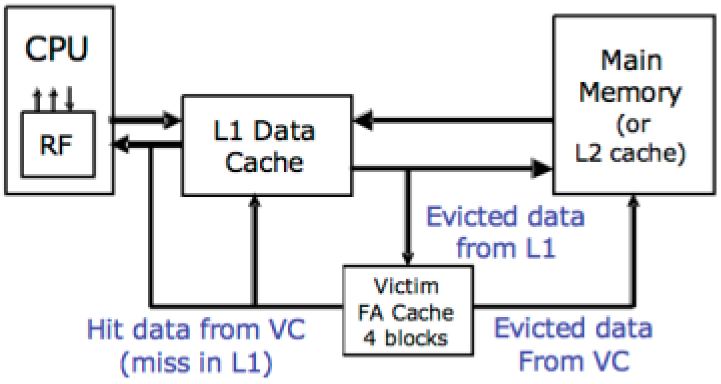
This constraint affects conflict miss.

Worse than an **ordinary 4-way set-associative cache**
Better than **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. 尽管直接映射缓存比集合关联缓存具有访问时间更短的优点，但由于缺乏关联性，它们有更多的冲突 miss。 为了减少这些冲突遗漏，Norm Jouppi 提出了受害者缓存，其中在直接映射的 L1 缓存中添加了一个小型的全关联备份缓存（称为受害者缓存），以容纳最近退出的缓存行。

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.

1.检查 L1 数据缓存。 如果它保存了请求的数据，则返回数据。

2.如果数据不在 L1 高速缓存中，则检查受害者高速缓存。 如果它保存了请求的数据，则将数据移到 L1 高速缓存中并发送回处理器。 从 L1 高速缓存中逐出的数据被放入受害者高速缓存中，并被放置在 FIFO 替换队列的末尾。

3.如果两个缓存都不保存数据，则从内存中检索数据，并将其放入 L1 缓存中。 如果 L1 高速缓存需要逐出旧数据以为新数据腾出空间，则将旧数据放入受害者高速缓存

中，并放置在 FIFO 替换队列的末尾。 需要从受害者缓存中腾出空间以腾出空间的数据都将写回到内存中或丢弃（如果未修改）。

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.

block size: 8 bytes 一行两个 4bytes 的 word.

Sets:4

Associativty :4

Tag :32-3 = 29

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).

请填写表 2.5-1 缓存的每个元素上延迟。 使用表 2.5-1 中计算的数据，计算通过此缓存的关键路径延迟（从设置输入地址到有效输出驱动器和适当的数据输出驱动器都输出有效数据的时间）。

Component	Delay equation (ps)	FA(ps)
Comparator	20×(# of tag bits) + 100	20*(32-3)+100=680
N-to-1 MUX	$50 \times \log_2 N + 100$	50*1+100=150
Buffer driver	200	200
AND gate	100	100
OR gate	$50 \times \log_2 N + 100$	200
Data output driver	50×(associativity) + 100	300
Valid output driver	100	100

Table 2.5-1: Delay of each cache component

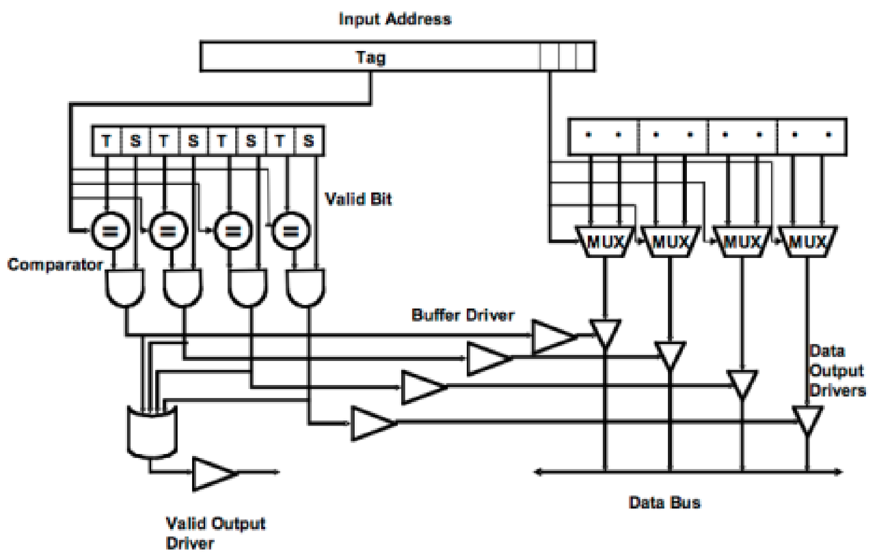


Figure 4: Victim cache datapath

Critical Path Cache Delay:
Tag check: comparator -> 2 in and -> 4in or -> Valid output driver
680+100+ 200+100 = 1080ps

comparator -> 2 in and ->Buffer driver-> Data output driver
680+100+ 200+300 = 1280ps

2-to-1 MUX - > Data output driver
150 + 300 = 450ps
So the critical path is comparator -> 2 in and ->Buffer driver-> Data output driver
the critical path delay through this cache 1280ps

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 4 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.
现在我们将研究受害者缓存对缓存命中率的影响。 我们的主要 L1 高速缓存是一个 128 字节的直接映射高速缓存，8 个 line ,一个 lines16 字节。 2 个 bits 寻址 word, 2 个 bits 寻址字节，图 4 中的牺牲者高速缓存是全关联高速缓存，2 个 line , 每条高速

缓存行具有 16 个字节，并且也是字可寻址的。 受害缓存使用先进先出（FIFO）替换策略。

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.
请完成表 2.5-2，其中显示了内存访问的痕迹。 在表中，每个条目均包含该行的标记，如果不存在数据，则包含 “ inv” 。 仅当值更改时，才应在表中填写元素。 为简单起见，地址仅为 8 位。 表格的前 3 行已为您填写。 为方便起见，下面描述了访问主缓存的地址明细。
 所以选择 L0,
选择 L0 ,把 0 evict 到 victim cache 中,

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			1						N			N
10		0							N			N
C0					1				N			N
18		0							Y			
20			0						N		A	N
8C	1								N	0		Y
28			0						Y			
AC			1						N		2	Y
38				0					N			N
C4					1				Y			
3C				0					Y			
48					0				N	C		N
0C	0								N		8	N
24			0						N	A		N

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?

0.15*50 = 7.5 cycles

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)	No effect.	Decrease, pingpong cache used to cause serious conflict misses problem but increasing associativity could solve it. By the way, most gain is from 1->2->4 way with limited benefit from higher associativity.	No effect, capacity does not change.
Halving the line size (associativity and # sets constant)	Increase due to spatial locality.	No effect due to constant blocks in cache.	Increase because the cache capacity is smaller.

Doubling the number of sets (capacity and line size constant)	No effect	Increase, because capacity constant, the number of associativity decrease,	No effect, capacity does not change.

	Compulsory Misses	Conflict Misses	Capacity Misses
Adding prefetching	No effect	Decrease, the processor detects the storage access mode of the running program, predicts which data will be accessed next time, and prefetches the data.	No effect

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)	Increase. We need mux correct way to processor. Smaller increases in hit time for further increases in associativity.	Decrease due to reduced conflict misses, but most gain is from 1->2->4 way with limited benefit from higher associativity.	No effect. Replacement policy runs in parallel with fetching missing line from memory.
Halving the line size (associativity and # sets constant)	No effect	Increase. It cannot use spatial locality better and increased capacity misses .	Decrease with smaller block size.
Doubling the number of sets (capacity and line size constant)	Decreases, tag has less bits, tag check can accelerate.	Increase, the associativity decrease because capacity constant.	No effect. line size constant

Adding prefetching	No effect	Decrease, the processor detects the storage access mode of the running program, predicts which data will be accessed next time, and prefetches the data.	No effect.