5 Solutions

Solution 5.1

5.1.1

a.	no solution provided
b.	no solution provided

5.1.2

a.	no solution provided
b.	no solution provided

5.1.3

a.	no solution provided
b.	no solution provided

5.1.4

a.	no solution provided
b.	no solution provided

5.1.5

a.	no solution provided
b.	no solution provided

5.1.6

a.	no solution provided
b.	no solution provided

Solution 5.2

5.2.1 4

5.2.2

a.	I, J
b.	B[I][0]

5.2.3

a.	A[I][J]
b.	A[J][I]

5.2.4

a.	3596 = 8 × 800/4 × 2 – 8 × 8/4 + 8000/4
b.	3186 = 8 × 800/4 × 2 – 8 × 8/4 + 8/4

5.2.5

a.	I, J
b.	I, J, B(I, 0)

5.2.6

a.	A(J, I)
b.	A(I, J), A(J, I), B(I, 0)

Solution 5.3

5.3.1

a.	no solution provided
b.	no solution provided

5.3.2

a.	no solution provided
b.	no solution provided

5.3.3

a.	No solution provided
b.	no solution provided

5.3.4

a.	no solution provided
b.	no solution provided

5.3.5 no solution provided

5.3.6 Yes, it is possible to use this function to index the cache. However, information about the six bits is lost because the bits are XOR'd, so you must include more tag bits to identify the address in the cache.

Solution 5.4

5.4.1

a.	8
b.	16

5.4.2

a.	32
b.	64

5.4.3

a.	1 + (22/8/32) = 1.086
b.	1 + (20/8/64) = 1.039

5.4.4 3

Address	0	4	16	132	232	160	1024	30	140	3100	180	2180
Line ID	0	0	1	8	14	10	0	1	9	1	11	8
Hit/miss	M	H	M	M	M	M	M	H	H	M	M	M
Replace	N	N	N	N	N	N	Y	N	N	Y	N	Y

5.4.5 0.25

5.4.6 <Index, tag, data>

<000001_2, 0001_2, mem[1024]>
<000001_2, 0011_2, mem[16]>
<001011_2, 0000_2, mem[176]>
<001000_2, 0010_2, mem[2176]>
<001110_2, 0000_2, mem[224]>
<001010_2, 0000_2, mem[160]>

Solution 5.5

5.5.1

a.	no solution provided
b.	no solution provided

5.5.2

a.	no solution provided
b.	no solution provided

5.5.3

a.	no solution provided
b.	no solution provided

5.5.4

a.	no solution provided
b.	no solution provided

5.5.5

a.	no solution provided
b.	no solution provided

5.5.6

a.	no solution provided
b.	no solution provided

Solution 5.6

5.6.1

no solution provided

5.6.2

no solution provided

5.6.3 With next-line prefetching, miss rate will be near 0%.

5.6.4

a.	no solution provided
b.	no solution provided

5.6.5

a.	no solution provided
b.	no solution provided

5.6.6

a.	no solution provided
b.	no solution provided

Solution 5.7

5.7.1

a.	P1	1.52 GHz
	P2	1.11 GHz
b.	P1	926 MHz
	P2	495 MHz

5.7.2

a.	P1	6.31 ns	9.56 cycles
	P2	5.11 ns	5.68 cycles
b.	P1	3.47 ns	3.21 cycles
	P2	4.07 ns	2.02 cycles

5.7.3

a.	P1	12.64 CPI	8.34 ns per inst	P2
	P2	7.36 CPI	6.63 ns per inst	
b.	P1	4.01 CPI	4.33 ns per inst	P1
	P2	2.38 CPI	4.81 ns per inst	

5.7.4

a.	6.50 ns	9.85 cycles	Worse
b.	3.84 ns	3.25 cycles	Worse

5.7.5

a.	13.04
b.	4.06

5.7.6 no solution provided

Solution 5.8

5.8.1

a.	no solution provided
b.	no solution provided

5.8.2

a.	no solution provided
b.	no solution provided

5.8.3

a.	no solution provided
b.	no solution provided

5.8.4

a.	no solution provided
b.	no solution provided

5.8.5

a.	no solution provided
b.	no solution provided

5.8.6

a.	no solution provided
b.	no solution provided

Solution 5.9

Instructors can change the disk latency, transfer rate, and optimal page size for more variants. Refer to Jim Gray's paper on the five-minute rule 10 years later.

5.9.1 32 KB.

5.9.2 Still 32 KB.

5.9.3 64 KB. Because the disk bandwidth grows much faster than seek latency, future paging cost will be closer to constant, thus favoring larger pages.

5.9.4 1987/1997/2007: 205/267/308 seconds (or roughly five minutes).

5.9.5 1987/1997/2007: 51/533/4935 seconds (or 10 times longer for every 10 years).

5.9.6 (1) DRAM cost/MB scaling trend dramatically slows down; or (2) disk $/access/sec dramatically increase. (2) is more likely to happen due to the emerging flash technology.

Solution 5.10

5.10.1

a.	no solution provided
b.	no solution provided

5.10.2

a.	no solution provided
b.	no solution provided

5.10.3

a.	no solution provided
b.	no solution provided

5.10.4

a.	no solution provided
b.	no solution provided

5.10.5

a.	no solution provided
b.	no solution provided

5.10.6

a.	no solution provided
b.	no solution provided

Solution 5.11

5.11.1

a.	no solution provided
b.	no solution provided

5.11.2

a.	no solution provided
b.	no solution provided

5.11.3

a.	no solution provided
b.	no solution provided

5.11.4 TLB initialization, or process context switch.

5.11.5 TLB miss. When most missed TLB entries are cached in processor caches.

5.11.6 Write protection exception.

Solution 5.12

5.12.1

a.	0 hits
b.	3 hits

5.12.2

a.	1 hit
b.	3 hits

5.12.3

a.	1 hit or fewer
b.	4 hits or fewer

5.12.4 5.12.4 Any address sequence is fine so long as the number of hits is correct.

a.	1 hit
b.	4 hits

5.12.5 The best block to evict is the one that will cause the fewest misses in the future. Unfortunately, a cache controller cannot know the future! Our best alternative is to make a good prediction.

5.12.6 If you knew that an address had limited temporal locality and would conflict with another block in the cache, it could improve miss rate. On the other hand, you could worsen the miss rate by choosing poorly which addresses to cache.

Solution 5.13

5.13.1 Shadow page table: (1) VM creates page table, hypervisor updates shadow table; (2) nothing; (3) hypervisor intercepts page fault, creates new mapping, and invalidates the old mapping in TLB; (4) VM notifies the hypervisor to invalidate the process's TLB entries. Nested page table: (1) VM creates new page table, hypervisor adds new mappings in PA to MA table; (2) hardware walks both page tables to translate VA to MA; (3) VM and hypervisor update their page tables, hypervisor invalidates stale TLB entries; (4) same as shadow page table.

5.13.2

Native: 4; NPT: 24 (instructors can change the levels of page table)

Native: L; NPT: $L \times (L + 2)$

5.13.3

Shadow page table: page fault rate

NPT: TLB miss rate

5.13.4

Shadow page table: 1.03

NPT: 1.04

5.13.5 Combining multiple page table updates

5.13.6 NPT caching (similar to TLB caching)

Solution 5.14

5.14.1

a.	no solution provided
b.	no solution provided

5.14.2

a.	no solution provided
b.	no solution provided

5.14.3 Virtual memory aims to provide each application with the illusion of the entire address space of the machine. Virtual machines aim to provide each operating system with the illusion of having the entire machine to its disposal. Thus they both serve very similar goals, and offer benefits such as increased security. Virtual memory can allow for many applications running in the same memory space to not have to manage keeping their memory separate.

5.14.4 Emulating a different ISA requires specific handling of that ISA's API. Each ISA has specific behaviors that will happen upon instruction execution, interrupts, trapping to kernel mode, etc. that therefore must be emulated. This can require many more instructions to be executed to emulate each instruction than was originally necessary in the target ISA. This can cause a large performance impact and make it difficult to properly communicate with external devices. An emulated system can potentially run faster than on its native ISA if the emulated code can be dynamically examined and optimized. For example, if the underlying machine's ISA has a single instruction that can handle the execution of several of the emulated system's instructions, then potentially the number of instructions executed can be reduced. This is similar to the recent Intel processors that do micro-op fusion, allowing several instructions to be handled by fewer instructions.

Solution 5.15

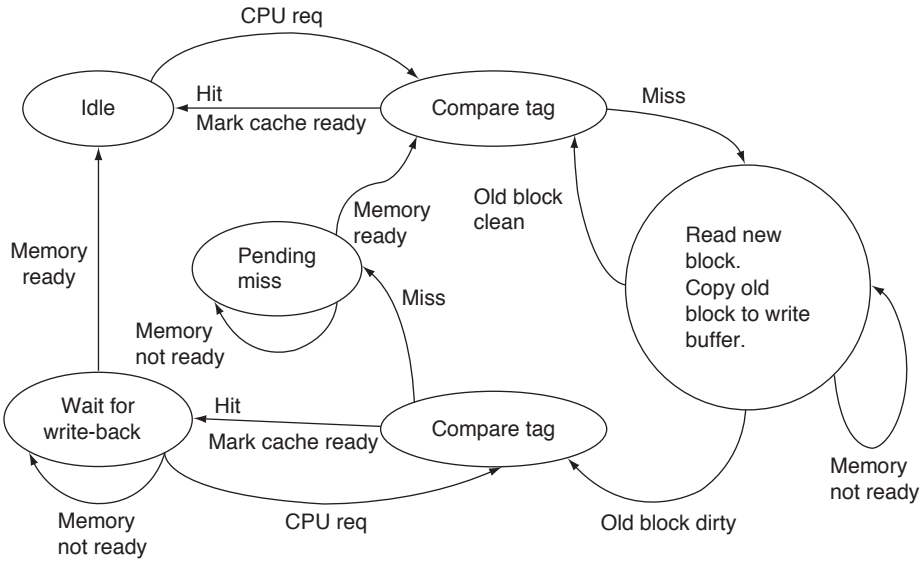
5.15.1 The cache should be able to satisfy the request since it is otherwise idle when the write buffer is writing back to memory. If the cache is not able to satisfy hits while writing back from the write buffer, the cache will perform little or no better than the cache without the write buffer, since requests will still be serialized behind writebacks.

5.15.2 Unfortunately, the cache will have to wait until the writeback is complete since the memory channel is occupied. Once the memory channel is free, the cache is able to issue the read request to satisfy the miss.

5.14.3 Correct solutions should exhibit the following features:

1. The memory read should come before memory writes.
2. The cache should signal "Ready" to the processor before completing the write.

Example (simpler solutions exist; the state machine is somewhat underspecified in the chapter):



Solution 5.16

5.16.1

a.	no solution provided
b.	no solution provided

5.16.2 no solution provided

5.16.3

a.	no solution provided
b.	no solution provided

5.16.4

a.	no solution provided
b.	no solution provided

5.16.5

a.	no solution provided
b.	no solution provided

5.16.6 Write-through, non-write allocate simplifies the most.

Solution 5.17

5.17.1

a.	no solution provided
b.	no solution provided

5.17.2

a.	no solution provided
b.	no solution provided

5.17.3

a.	no solution provided
b.	no solution provided

5.17.4

a.	no solution provided
b.	no solution provided

5.17.5

a.	no solution provided
b.	no solution provided

5.17.6

Processor: out-of-order execution, larger load/store queue, multiple hardware threads

Caches: more miss status handling registers (MSHR)

Memory: memory controller to support multiple outstanding memory requests

Solution 5.18

5.18.1

a.	`srcIP` and `refTime` fields. 2 misses per entry.
b.	`srcIP` and `browser` fields. 1 miss per entry.

5.18.2

a.	Group the `srcIP` and `refTime` fields into a separate array.
b.	Split the `srcIP` into a separate array; have a hash table on the browser field.

5.18.3

a.	`peak_hour (int status); // peak hours of a given status`
	Group `srcIP`, `refTime`, and `status` together.

b.	`topK_sourceIP (int hour);` Group the `srcIP` and `refTime` fields into a separate array, and a browser hash table.

5.18.4

a.	no solution provided
b.	no solution provided

5.18.5

a.	no solution provided
b.	no solution provided

5.18.6

a.	apsi/mesa/ammp/mcf all have such examples.
b.	apsi/mesa/ammp/mcf all have such examples.

Example cache: 4-block caches, direct-mapped vs. 2-way LRU.

Reference stream (blocks): 1 2 2 6 1.