Chapter 3

1. Given the following benchmark code, and assuming a virtually-addressed fully-associative cache with infinite capacity and 64 byte blocks, compute the overall miss rate (number of misses divided by number of references). Assume that all variables except array locations reside in registers, and that arrays A, B, and C are placed consecutively in memory.

```
double A[1024], B[1024], C[1024];
for(int i=0;i<1000;i += 2) {
    A[i] = 35.0 * B[i] + C[i+1];
}
```

   There are 3000 memory references (1000 each to A, B, C)

   For each array, these span ceil(1000*8/64) = 128 blocks, hence there are 128 misses to each array, leading to 3x128 = 384 total misses. The total miss rate is 384/3000 = 12.8% misses per reference.

2. Given the example code in Problem 1, and assuming a virtually-addressed direct-mapped cache of capacity 8KB and 64 byte blocks, compute the overall miss rate (number of misses divided by number of references). Assume that all variables except array locations reside in registers, and that arrays A, B, and C are placed consecutively in memory.

   The first iteration accesses memory location &B[0], &C[1], and &A[0]. Unfortunately, since the arrays are consecutive in memory, these locations are exactly 8KB (1024 x 8B per double) apart. Hence, in a direct-mapped cache they conflict, and the access to C[1] will evict B[0], while the access to A[0] will evict C[1]. As a result, every reference will miss, leading to a 100% miss rate.

3. Given the example code in Problem 1, and assuming a virtually-addressed two-way set associative cache of capacity 8KB and 64 byte blocks, compute the overall miss rate (number of misses divided by number of references). Assume that all variables except array locations reside in registers, and that arrays A, B, and C are placed consecutively in memory.

   The first iteration accesses memory location &B[0], &C[1], and &A[0]. Unfortunately, since the arrays are consecutive in memory, these locations are exactly 8KB (1024 x 8B per double) apart. Hence, in a two-way set-associative cache they conflict, and the access to A[0] will evict B[0]. In the second iteration, the access to B[1] will evict C[1], and so on. However, since the access to C is offset by 1 double (8 bytes), in the seventh iteration it will access C[8], which does not conflict with B[7]. Hence, B[7] will hit, as will A[7]. In the eighth iteration, C[9] will also hit, but now B[8] and A[8] will again conflict, and no hits will result. Hence, there are three hits

every eight iterations, leading to a total number of hits of floor(1000/8)*3 = 375 hits. The number of misses is 3000-375 = 2625, for an overall miss rate of 87.5% misses per reference.

4. Consider a cache with 256 bytes. Word size is 4 bytes and block size is 16 bytes. Show the values in the cache and tag bits after each of the following memory access operations for the two cache organizations direct mapped and 2-way associative. Also indicate whether the access was a hit or a miss. Justify. The addresses are in hexadecimal representation. Use LRU (least recently used) replacement algorithm wherever needed.

   1. Read 0010
   2. Read 001C
   3. Read 0018
   4. Write 0010
   5. Read 0484
   6. Read 051C
   7. Read 001C
   8. Read 0210
   9. Read 051C

Reference:	00**1**0	00**1**C	00**1**8	00**1**0	04**8**4	05**1**C	00**1**C	02**1**0	05**1**C
Hit/miss:	M	H	H	H	M	M	M	M	M
Direct-mapped cache tag state									
Set 0									
Set 1	00	00	00	00	00	05	00	02	05
Set 2									
Set 3									
Set 4									
Set 5									
Set 6									
Set 7									
Set 8					04	04	04	04	04
Set 9									
Set A									
Set B									
Set C									
Set D									
Set E									
Set F									

Reference:	00**1**0	00**1**C	00**1**8	00**1**0	04**8**4	05**1**C	00**1**C	02**1**0	05**1**C
Hit/miss:	M	H	H	H	M	M	H	M	M
Two-way set-associative tag state									
Set 0					04	04	04	04	04
Set 1	00	00	00	00	00	05	00	02	05
						00	05	00	02
Set 2									
Set 3									
Set 4									
Set 5									
Set 6									
Set 7									

5. Describe a program that has very high temporal locality. Write pseudocode for such a program, and show that it will have a high cache hit rate.

   Solution not provided.

6. Describe a program that has very low temporal locality. Write pseudocode for such a program, and show that it will have a high cache miss rate.

   Solution not provided.

7. Write the programs of Problem 5 and Problem 6 and compile them on a platform that supports performance counters (for example, Microsoft Windows and the Intel VTune performance counter software). Collect and report performance counter data that verifies that the program with high temporal locality experiences fewer cache misses.

   Solution not provided.

8. Write the programs of Problem 5 and Problem 6 in C, and compile them using the Simplescalar compilation tools (gcc, ld) available from http://www.simplescalar.com. Download and compile the Simplescalar 3.0 simulation suite and use the sim-cache tool to run both programs.Verify that the program with high temporal locality experiences fewer cache misses by reporting cache miss rates from both programs.

   Solution not provided.

9. Describe a program that has very high spatial locality. Write pseudocode for such a program, and show that it will have a high cache hit rate.

   Solution not provided.

10. Describe a program that has very low spatial locality. Write pseudocode for such a program, and show that it will have a high cache miss rate.

    Solution not provided.

11. Write the programs of Problem 9 and Problem 10 and compile them on a platform that supports performance counters (for example, Linux and the Intel VTune performance counter software). Collect and report performance counter data that verifies that the program with high temporal locality experiences fewer cache misses.

    Solution not provided.

12. Write the programs of Problem 9 and Problem 10 in C, and compile them using the Simplescalar compilation tools (gcc, ld) available from http://www.simplescalar.com. Download and compile the Simplescalar 3.0 simulation suite and use the sim-cache tool to run both programs.Verify that the program with high temporal locality experiences fewer cache misses by reporting cache miss rates from both programs.

    Solution not provided.

13. Consider a processor with 32-bit virtual addresses, 4KB pages and 36-bit physical addresses. Assume memory is byte-addressable (i.e. the 32-bit VA specifies a byte in memory).

    ❑L1 instruction cache: 64 Kbytes, 128 byte blocks, 4-way set associative, indexed and tagged with virtual address.

    ❑L1 data cache: 32 Kbytes, 64 byte blocks, 2-way set associative, indexed and tagged with physical address, write-back.

    ❑4-way set associative TLB with 128 entries in all. Assume the TLB keeps a dirty bit, a reference bit, and 3 permission bits (read, write, execute) for each entry.

    Specify the number of offset, index, and tag bits for each of these structures in the table below. Also, compute the total size in number of bit cells for each of the tag and data arrays.

Structure	Offset bits	Index bits	Tag bits	Size of tag array	Size of data array
I-cache	7	7	18	512x(18 tag + 1 valid) = 9728 bits	64KB
D-cache	6	8	22	512x(22+1 valid + 1 dirty) = 12,288 bits	32KB
TLB	12	5	15	128x(15 tag + v) = 2048 bits	128x(24+d+r +3p) = 3712 bits

14. Given the cache organization in Problem 13, explain why accesses to the data cache would take longer than accesses to the instruction cache. Suggest a lower-latency data cache design with the same capacity and describe how the organization of the cache would have to change to achieve the lower latency.

    This design serializes TLB and data cache lookup, since the data cache is physically addressed. Two alternatives exist: a virtually-addressed data cache, or a virtually-indexed physically-tagged data cache. The latter would have to be 32KB/4KB = 8-way set associative to avoid using any translated bits in the index.

15. Given the cache organization in Problem 13, assume the architecture requires writes that modify the instruction text (i.e. self-modifying code) to be reflected immediately if the modified instructions are fetched and executed. Explain why it may be difficult to support this requirement with this instruction cache organization.

    The virtually-addressed instruction cache would have to be checked, or snooped, for every store. Since the data cache is physically addressed, a separate address path would have to be provided that uses the store's virtual address to snoop the instruction cache. Furthermore, virtual-address aliases could cause complications, since the instruction cache may store a block with one virtual address, while the program performs a write with a different virtual address.

16. Assume a two-level cache hierarchy with a private level one instruction cache (L1I), a private level one data cache (L1D), and a shared level two data cache (L2). Given local miss rates for the 4% for L1I, 7.5% for L1D, and 35% for L2, compute the global miss rate for the L2 cache.

    L2 global miss rate = (.35 L2 misses)/(L2 ref) x ((.04 L2 ref)/Ifetch + (.075 L2 ref) / D-ref) = .35x.04 + .35x.075 = 0.04025 = 4.025% L2 misses per global reference

17. Assuming 1 L1I access per instruction and 0.4 data accesses per instruction, compute the misses per instruction for the L1I, L1D, and L2 caches of Problem 16.

    L1 I misses per instruction = .04 miss/ref x 1 ref/instr = .04 miss/inst

    L1 D misses per instruction = .075 miss/ref x 0.4 ref/instr = .03 miss/instr

    L2 misses per instruction = .35 miss/ref x (.04 ref/inst + .03 ref/inst) = .35 x .07 = .0245 miss/inst

18. Given the miss rates of Problem 16, and assuming that accesses to the L1I and L1D caches take one cycle, accesses to the L2 take 12 cycles, accesses to main memory take 75 cycles, and a clock rate of 1GHz, compute the average memory reference latency for this cache hierarchy.

    Avg Ifetch mem lat = (1-.04) x 1 + .04 x (1-.35) x 12 + .04 x.35 x 75 = .96 + .312 +1.05 = 2.322 cycles

Avg data ref mem lat = (1-.075) x 1 + .075 x (1-.35) x 12 + .075 x.35 x 75 = .925 + .567 + 1.96875 = 3.46075 cycles

NOTE: to determine overall average memory latency, you must know the ratio of data and instruction references and take the weighted average of the two types of references. As in Problem 17, assume 0.4 data references per instruction reference:

Avg mem lat = (2.322 cycles / ifetch x (1 ifetch) / ref + 3.46075 cycles/dref x (0.4 dref/ref)) / 1.4 = 2.647 cycles

19. Assuming a perfect cache CPI (cycles per instruction) for a pipelined processor equal to 1.15 CPI, compute the MCPI and overall CPI for a pipelined processor with the memory hierarchy described in Problem 18 and the miss rates and access rates specified in Problem 16 and Problem 17.

From solution to 18, miss rates per instruction are .04, .03, and .0245, hence:

MCPI = .04 x 12 + .03 x 12 + .0245 x (75 - 12) = 2.3835

CPI = 1.15 + MCPI = 1.15 + 2.3835 + 3.5335

20. Repeat Problem 16 assuming an L1I local miss rate of 7%, an L1D local miss rate of 3.5%, and an L2 local miss rate of 75%.

Solution not provided (plug into equations of solution to Problem 16).

21. Repeat Problem 17 given the miss rates of Problem 20.

Solution not provided (plug into equations of solution to Problem 17).

22. Repeat Problem 18 given the miss rates of Problem 20.

Solution not provided (plug into equations of solution to Problem 18).

23. Repeat Problem 19 given the miss rates of Problem 20.

Solution not provided (plug into equations of solution to Problem 19).

24. CPI equations can be used to model the performance of in-order superscalar processors with multilevel cache hierarchies. Compute the CPI for such a processor, given the following parameters:

- Infinite cache CPI of 1.15
- L1 cache miss penalty of 12 cycles

- L2 cache miss penalty of 50 cycles
- L1 instruction cache per-instruction miss rate of 3% (0.03 misses/instruction)
- L1 data cache per-instruction miss rate of 2% (0.02 misses/instruction).
- L2 local cache miss rate of 25% (0.25 misses/L2 reference).

CPI = 1.15 + 12 cycles/miss x (.03 miss/inst + .02 miss/inst) + 50 cycles/l2miss x (0.25 l2miss/l2ref x (.03+.02) l2ref/inst)

= 1.15 + 0.60 + 0.625 = 2.375 cyc/inst

25. It is usually the case that a set-associative or fully-associative cache has a higher hit rate than a direct-mapped cache. However, this is not always true. To illustrate this, show a memory reference trace for a program that has a higher hit rate with a 2-block direct-mapped cache than a fully-associative cache with 2 blocks.

    Assume A and B map to the same set but C maps to the other set:

    A,B,C,A,B,C,A,B,C,... would always miss in the fully-associative cache, resulting in a 100% miss rate

    However, in a direct-mapped cache only A and B would always miss, but C would remain resident, resulting in a 67% miss rate.

26. Download and install the Simplescalar 3.0 simulation suite and instructional benchmarks from www.simplescalar.com. Using the sim-cache cache simulator, plot the cache miss rates for each benchmark for the following cache hierarchy: 16KB 2-way set-associative L1 instruction cache with 64 byte lines; 32KB 4-way set-associative L1 data cache with 32 byte lines; 512KB 8-way set-associative L2 cache with 64 byte lines.

    Solution not provided.

27. Using the benchmarks and tools from Problem 26, plot several miss-rate sensitivity curves for each of the three caches (L1I, L1D, L2) by varying each of the following parameters: cache size 0.5x, 1x, 2x, 4x; associativity 0.5x, 1x, 2x, 4x; block size 0.25x, 0.5x, 1x, 2x, 4x. Hold the other parameters fixed at the values in Problem 26 while varying each of the three parameters for each sensitivity curve. Based on your sensitivity curves, identify an appropriate value for each parameter near the knee of the curve (if any) for each benchmark.

    Solution not provided.

28. Assume a synchronous front-side processor-memory bus that operates at 100MHz and has an 8-byte data bus. Arbitration for the bus takes one bus cycle (10ns), issuing a cache line read command for 64 bytes of data takes one cycle, memory controller latency (including DRAM access) is 60ns, after which data doublewords are returned in back-to-back cycles. Further assume the

bus is blocking or circuit-switched. Compute the latency to fill a single 64 byte cache line. Then compute the peak read bandwidth for this processor-memory bus, assuming the processor arbitrates for the bus for a new read in the bus cycle following completion of the last read.

Fill latency = arb + cmd + mem + #transfers = 1 + 1 + 60/10 + (64/8) = 16 bus cycles or 160ns

Peak read bandwidth = 64B/160ns = 0.4B/ns = 400 million bytes/sec.

29. Given the assumptions of Problem 28, assume a nonblocking (split-transaction) bus that overlaps arbitration with commands and data transfers, but multiplexes data and address lines. Assume that a read command requires a single bus cycle, and further assume that the memory controller has infinite DRAM bandwidth. Compute the peak data bandwidth for this front side bus.

    Arbitration no longer consumes a bus cycle. However, there is still a read on the bus (since address and data lines are multiplexed) for every 8 transfers. Hence, 8 of 9 bus cycles transfer data in the best (peak) case, so the bus could sustain:

    Peak data bandwidth = 8B / cycle x 1 cycle/10ns x 8/9 = 711 million bytes/sec

30. Building on the assumptions of Problem 29, assume the bus now has dedicated data lines and a separate arbitration mechanism for addresses/commands and data. Compute the peak data bandwidth for this front side bus.

    In this case, arbitration no longer consumes a bus cycle, and neither do read commands. Hence, the peak bandwidth is:

    Peak data bandwidth = 8B /cycle x 1 cycle/10 ns = 800 million bytes/sec

31. Consider finite DRAM bandwidth at a memory controller, as follows. Assume double-data-rate DRAM operating at 100 MHz in a parallel non-interleaved organization, with an 8 byte interface to the DRAM chips. Further assume that each cache line read results in a DRAM row miss, requiring a precharge and RAS cycle, followed by row-hit CAS cycles for each of the doublewords in the cache line. Assuming memory controller overhead of one cycle (10ns) to initiate a read operation, and one cycle latency to transfer data from the DRAM data bus to the processor-memory bus, compute the latency for reading one 64 byte cache block. Now compute the peak data bandwidth for the memory interface, ignoring DRAM refresh cycles.

    Memory latency is measured from when the memory controller sees the command to when it places the last doubleword on the processor bus:

    Latency = precharge + RAS + overhead + 64Bx(1 xfer/8B)x(1 cycle)/(2 xfer)) = 7 cycles = 70ns

    Peak data bandwidth = 64B/70ns = 0.914B/ns = 914 million bytes/sec

32. Two page-table architectures are in common use today: multilevel forward page tables and hashed page tables. Write out a pseudocode function matching the following function declaration that searches a 3-level forward page table and returns 1 on a hit and 0 on a miss, and assigns *realaddress on a hit.

```
int fptsearch(void *pagetablebase, void* virtualaddress, void**
realaddress) {
#define L1BITS 6
#define L2BITS 7
#define L3BITS 7
#define PAGEOFFSETBITS 12
unsigned int l1offset = virtualaddress >> (32 - L1BITS)
if (pagetablebase[l1offset].valid == 0)
  return 0;
void *level2 = pagetablebase[l1offset].ptr;
unsigned int l2offset = (virtualaddress << L1BITS) >> (32 -
L2BITS);
if (level2[l2offset].valid == 0)
  return 0;
void *level3 = level2[level2offset];
unsigned int l3offset = (virtualaddress << (L1BITS +
L2BITS))>>(32-PAGEOFFSETBITS);
if (level3[l3offset].valid == 0)
  return 0;
*realaddress = level3[l3offset].realaddress;
return 1;
}
```

33. As in Problem 32, write out a pseudocode function matching the following function declaration that searches a hashed page table and returns 1 on a hit and 0 on a miss, and assigns *realaddress on a hit:

```
int hptsearch(void *pagetablebase, void* virtualaddress, void**
realaddress) {
#define PTEPERGROUP 4
void *hash = pagetablebase[ptehash(virtualaddress)];
for(int i=0; i< PTEPERGROUP; ++i) {
  if (hash[i].valid && (hash[i].virtualaddress == virtualad-
dress)) {
    *realaddress = hash[i].realaddress;
     return 1;
  }
```

```
}
return 0;
}
```

34. Assume a single-platter disk drive with average seek time of 4.5ms, rotation speed of 7200RPM, data transfer rate of 10MB/s per head, and controller overhead and queueing of 1ms. What is the average access latency for a 4096 byte read?

```
latency = rot. latency + seek latency + transfer latency + over-
head
 = (1 min/7200 rev x 60000ms / min x .5 rot/access) + 4.5ms +
4096B/10MB/s + 1
 = 4.17ms + 4.5ms + .4ms + 1ms = 9.67ms
```

35. Recompute the average access latency for Problem 34 assuming a rotation speed of 15K RPM, two platters, and an average seek time of 4.0ms.

```
latency = rot. latency + seek latency + transfer latency + over-
head
 = (1 min/15000 rev x 60000ms / min x .5 rot/access) + 4.0ms +
4096B/2/10MB/s + 1
 = 2ms + 4.0ms + .2ms + 1ms = 7.3ms
```