Exercises Week 11
Textbook Sections 5.7, 5.8, 5.13
Sample Solutions

Question 1

Textbook Exercise 5.16 <§5.7>

As described in Section 5.7, virtual memory uses a page table to track the mapping of virtual addresses to physical addresses. This exercise shows how this table must be updated as addresses are accessed. The following table is a stream of virtual byte addresses as seen on a system. Assume 4 KiB pages, a four-entry fully-associative TLB, and true LRU replacement. If pages must be brought in from disk, increment the next largest page number [i.e., use the next largest physical page number beyond those already allocated].

Decimal	4669	2227	13916	34587	48870	12608	49225
Hex	0x123d	0x08b3	0x365c	0x871b	0xbee6	0x3140	0xc049

TLB initial state:

Valid	Tag	Physical Page Number	Time Since Last Access
1	0xb	12	4
1	0x7	4	1
1	0x3	6	3
0	0x4	9	7

Page table initial state:

Index	Valid	Physical page or in disk
0	1	5
1	0	Disk
2	0	Disk
3	1	6
4	1	9
5	1	11
6	0	Disk
7	1	4
8	0	Disk
9	0	Disk
a	1	3
b	1	12

a. For each access shown above, list

- whether the access is a hit or miss in the TLB,
- whether the access is a hit or miss in the page table,
- whether the access is a page fault,
- the updated state of the TLB.

With 4KiB pages, the least-significant 12 bits of a virtual address are the byte offset within a page, and the remaining most-significant bits are the virtual page number. This conveniently divides an address

expressed in hexadecimal into the rightmost three digits for the byte offset and the remaining leftmost digits as the virtual page number.

The trace of TLB and page table lookups is:

- VA 0x123d: VPN 0x1
  TLB: miss, replace entry 3
  Page table: page fault, allocate physical page 13

Valid	Tag	Physical Page Number	Time Since Last Access
1	0xb	12	5
1	0x7	4	2
1	0x3	6	4
1	0x1	13	0

- VA 0x08b3: VPN 0x0
  TLB: miss, replace entry 0
  Page table: hit, physical page 5

Valid	Tag	Physical Page Number	Time Since Last Access
1	0x0	5	0
1	0x7	4	3
1	0x3	6	5
1	0x1	13	1

- VA 0x365c: VPN 0x3
  TLB: hit, physical page 6

Valid	Tag	Physical Page Number	Time Since Last Access
1	0x0	5	1
1	0x7	4	4
1	0x3	6	0
1	0x1	13	2

- VA 0x871b: VPN 0x8
  TLB: miss, replace entry 1
  Page table: page fault, allocate physical page 14

Valid	Tag	Physical Page Number	Time Since Last Access
1	0x0	5	2
1	0x8	14	0
1	0x3	6	1
1	0x1	13	3

- VA 0xbee6: VPN 0xb
  TLB: miss replace entry 3
  Page table: hit, physical page 12

Valid	Tag	Physical Page Number	Time Since Last Access
1	0x0	5	3
1	0x8	14	1
1	0x3	6	2
1	0xb	12	0

- VA 0x3140: VPN 0x3
  TLB: hit, physical page 6

Valid	Tag	Physical Page Number	Time Since Last Access
1	0x0	5	4
1	0x8	14	2
1	0x3	6	0
1	0xb	12	1

- VA 0xc049: VPN 0xc
  TLB: miss
  Page table: page fault, illegal virtual address

Valid	Tag	Physical Page Number	Time Since Last Access
1	0x0	5	5
1	0x8	14	3
1	0x3	6	1
1	0xb	12	2

Final Page Table:

Virtual Page No.	Valid	Physical page or in disk
0	1	5
1	1	13
2	0	Disk
3	1	6
4	1	9
5	1	11
6	0	Disk
7	1	4
8	1	14
9	0	Disk
10	1	3
11	1	12

Question 2

Textbook Exercise 5.17 <§5.7>

There are several parameters that impact the overall size of the page table. Listed below are several key page table parameters.

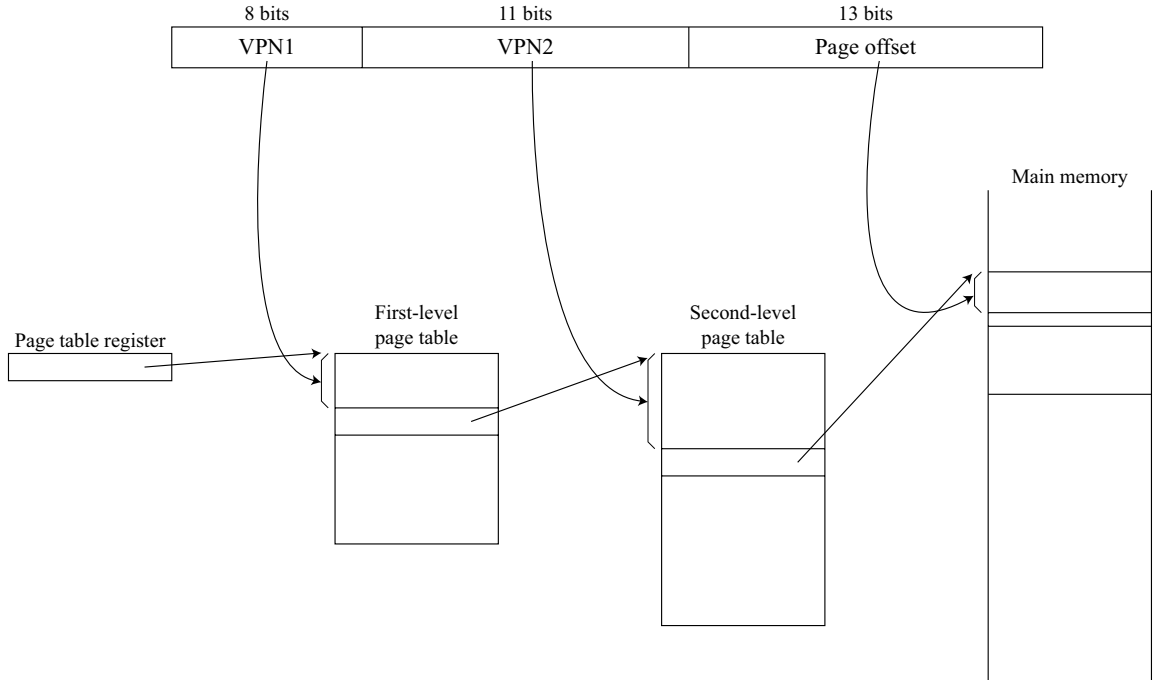
Virtual address size	Page size	Page table entry size
32 bits	8 KB	4 bytes

a. Given the parameters shown above, calculate the maximum possible page table size for a system running five processes.

32-bit addresses give 2^{32} bytes of virtual address space. Divide this into 8KB (2^{13} byte) pages, so there are $2^{32-13} = 2^{19}$ pages (i.e., 512K pages) in a full address space, each requiring a PTE. Each PTE is 4 bytes, so the page table for each full address space is 2MB. Assuming each application has its own address space, the page tables for five applications require 10MB.

b. Given the parameters shown above, calculate the total page table size for a system running five applications that each utilize half of the virtual memory available, given a two-level page table approach with up to 256 entries at the first level. Assume each entry of the main page table is 6 bytes. Calculate the minimum and maximum amount of memory required for this page table.

The multi-level page table scheme is outlined as option 5 in the subsection titled "Virtual Memory for Large Virtual Addresses" in Section 5.7 on page 429. If there are 256 entries in the first-level table, the virtual page number (VPN) of a virtual address is subdivided into VPN1 and VPN2 fields as shown below.



The first-level page table contains 256 entries of 6 bytes each (1.5Kbytes total) and provides the first step of translation for 16Mbyte segments of address space. If a program does not use any of the addresses in a segment, the corresponding first-level PTE is marked invalid, and no second-level page table is required for the segment.

On the other hand, if any addresses in a segment are used, the first-level PTE points to a second-level page table for the segment. A second-level page table has 2^{11} = 2K entries, each of 6 bytes (12Kbytes total). Each second-level PTE, if valid, points to an 8Kbyte page in physical memory, just as in the single-level scheme.

For an application using half the virtual memory available, the total page table size depends on how the pages used are distributed amongst the 16Mbyte segments. The minimum page table size occurs if the pages are consolidated into half the available segments, since then no second-level page tables are needed for the other half of the available segments. There are 256 segments available, so if only half are used, we only need 128 second-level page tables, each of 12Kbytes. We always need the first-level page table of 1.5Kbytes. So the storage for page tables for each application is 1.5Kbytes + 128 × 12Kbytes = 1537.5Kbytes, or approximately 1.5Mbytes. For 5 such applications, the total is approximately 7.5Mbytes.

The maximum page table size occurs if the pages are distributed amongst all of the available 16Mbyte segments (e.g., half of each segment is used). In that case, we need 256 second-level page tables, each of 12Kbytes, plus the first-level page table of 1.5Kbytes. The storage for page tables for each application is 1.5Kbytes + 256 × 12Kbytes = 3073.5Kbytes. For 5 such applications, the total is 15367.5Kbytes, or approximately 15Mbytes.

Question 3

Many microprocessors use the AMBA bus to interconnect processors and memory within a chip. The AMBA specification includes features to support cache coherence, based on a common cache block size of 64 bytes used by all of the components in the system. While that relieves a cache designer of one design decision, there are still choices to make for cache size and associativity.

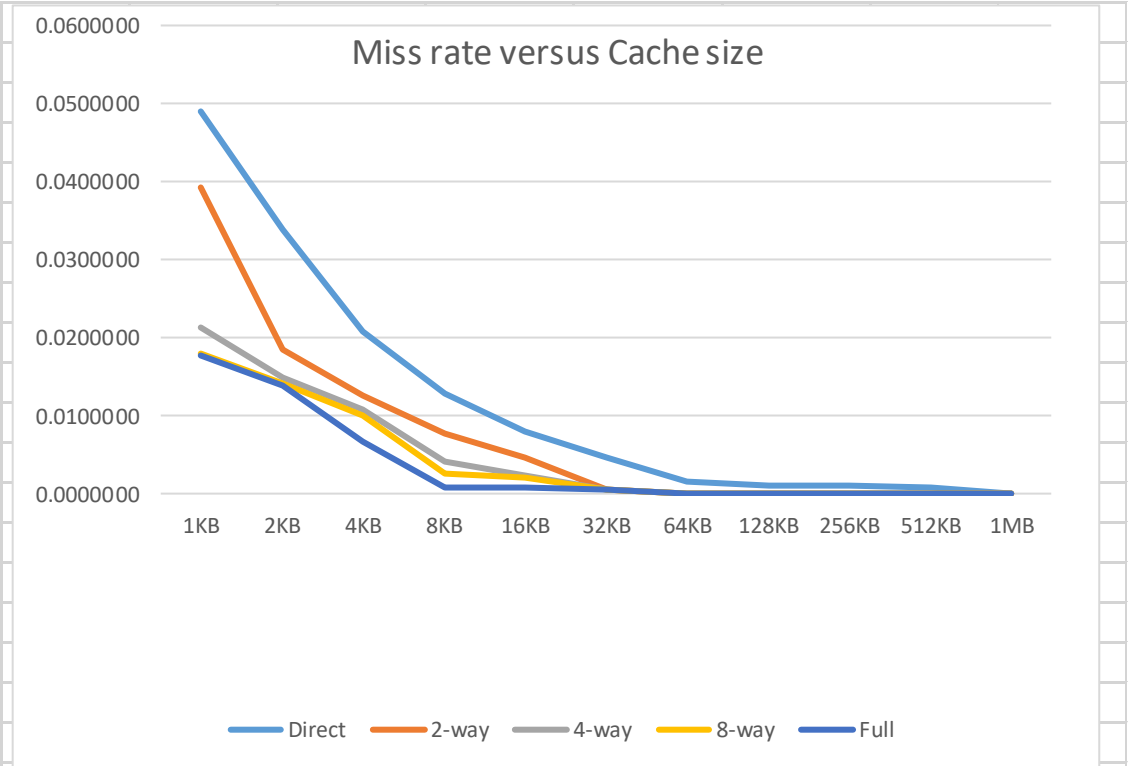
The spreadsheet dcache.xlsx provided for this exercise (shown below) has miss-rate data measured from simulations of data caches of various sizes and associativity, for a processor running the SPEC-2000 benchmark suite (Cantin and Hill, "Cache Performance for SPEC CPU2000 Benchmarks," May 2003, http://research.cs.wisc.edu/multifacet/misc/spec2000cache-data/).

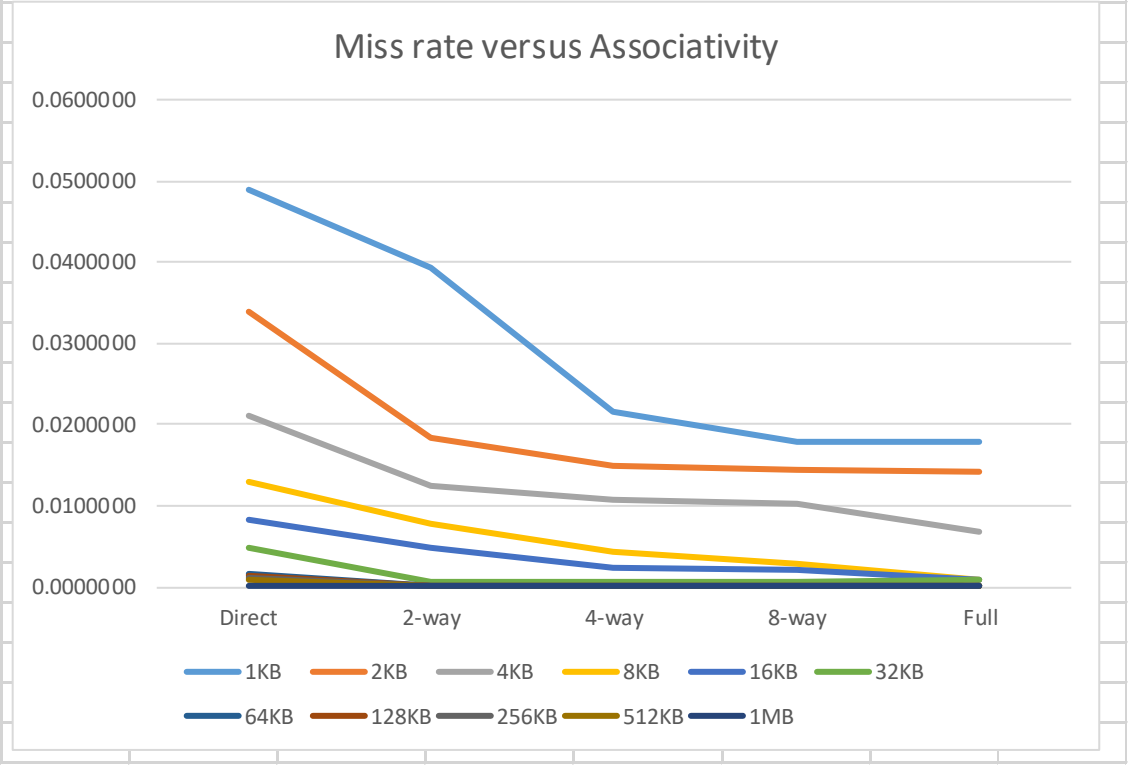
Size	Direct	2-way	4-way	8-way	Full
1KB	0.0490226	0.0392158	0.0214944	0.0179407	0.0177980
2KB	0.0338158	0.0184496	0.0149315	0.0143292	0.0140771
4KB	0.0209587	0.0125664	0.0108321	0.0101989	0.0067031
8KB	0.0130101	0.0078696	0.0043419	0.0027407	0.0009342
16KB	0.0081525	0.0048237	0.0024620	0.0020958	0.0008781
32KB	0.0047908	0.0006324	0.0006225	0.0007058	0.0007660
64KB	0.0017175	0.0001342	0.0000687	0.0000570	0.0000425
128KB	0.0012558	0.0000514	0.0000402	0.0000366	0.0000363
256KB	0.0011399	0.0000394	0.0000362	0.0000361	0.0000361
512KB	0.0009755	0.0000389	0.0000360	0.0000360	0.0000359
1MB	0.0000465	0.0000383	0.0000353	0.0000352	0.0000351

Use Excel or other spreadsheet or graphing tools to create two multi-line graphs:

- Miss rate versus associativity, for various cache sizes
- Miss rate versus cache size, for various degrees of associativity

Comment on your observations from these visualizations. What guidance does this data offer the cache designer?





Clearly the best miss rates are obtained with a large cache and high degreee of associativity. Increasing associativity beyond a certain amount appears to provide diminishing returns. For larger caches, the associativity beyond which miss rate reduces appreciably is less than for smaller caches. This suggests that a large cache can have smaller associativity, and a small cache should have higher associativity. That is what we see in real systems, and is probably based on data like this.