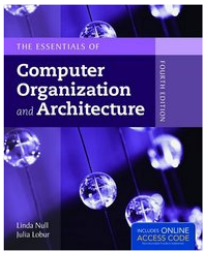


The Essentials of Computer Organization and Architecture
4th Edition

Linda Null and Julia Lobur
Jones and Bartlett Publishers © 2015

Chapter 6 Instructor's Manual

Chapter Objectives

Chapter 6, Memory, covers basic memory concepts, such as RAM and the various memory devices, and also addresses the more advanced concepts of the memory hierarchy, including cache memory and virtual memory. This chapter gives a thorough presentation of direct mapping, associative mapping, and set-associative mapping techniques for cache. It also provides a detailed look at overlays, paging and segmentation, TLBs, and the various algorithms and devices associated with each. A tutorial and simulator for this chapter is available.

Lectures should focus on the following points:

- **Types of memory.** There are many types of memory, but the two basic categories are RAM and ROM.
- **The memory hierarchy.** One of the most important considerations in understanding the performance capabilities of a modern computer is the memory hierarchy. The goal of this section is understanding how system memory (registers, cache, and main memory), online memory (hard disk), near line memory (optical disk), and offline memory (tapes and floppy disks) work together to provide acceptable performance at a minimal cost. Locality of reference (or the clustering of memory references) is integral in understanding how a memory hierarchy works.
- **Cache memory.** The purpose of cache is to speed up memory accesses by storing recently used data closer to the CPU (in a memory that requires less access time). It is important to discuss where this data is stored in cache, so direct mapping, fully associative cache, and set associative cache are covered. The effective access time is a good way to measure the performance of cache.
- **Virtual memory.** Virtual memory is a method used to increase the available address space for a process by using the hard disk as an extension of RAM. Both paging and segmentation

(including advantages and disadvantages) are covered. In addition TLBs are introduced as a method for improving performance of paging systems.

- **Real-world examples of memory management.** The chapter concepts are studied in the context of the Pentium memory hierarchy.
-

Required Lecture Time

The important concepts in Chapter 6 can typically be covered in 3-5 lecture hours. However, if a teacher wants the students to have a mastery of all topics in Chapter 6, 8 lecture hours are more reasonable. If lecture time is limited, we suggest that the focus be on the memory hierarchy and cache memory. Virtual memory is often covered in an operating systems course, but we provide coverage in this textbook because we feel it is important that students see the hierarchy in its entirety.

Lecture Tips

Students often have difficulty understanding why the memory hierarchy works. In particular, many miss the concept of bringing in an entire block when a "miss" occurs (thus using the principle of locality). Instructors should focus on several small examples to make sure students understand the concept of locality. This will also help with the section on paging.

Although the cache mapping schemes are relatively straight forward, students have a tendency to miss *why* the schemes are necessary. (For example, many can work examples involving the various mapping schemes, but they aren't sure exactly what they are doing and why they are doing it.) It is important to stress not only how the main memory address is mapped to a cache location but why.

For the various cache mapping schemes, we have found that working out small examples in detail is a good way to approach these concepts. Examples showing blocks of memory actually put into a small cache, and then that cache getting full (or several blocks mapping to the same location in cache) are very helpful for motivating the need for the tag in cache.

If students are comfortable with cache mapping, paging is much easier for them to deal with, as many concepts are similar. However, these similarities can cause confusion for the students, as some tend to mix up cache and paging. Covering examples using both cache and paging helps to clear up this confusion.

Instructors need to differentiate between byte addressing and word addressing. If an architecture has 4-byte words, it can still be byte addressable. The exercises and problems in the book generally use byte addressing.

Answers to Exercises

1. Suppose a computer using direct mapped cache has 2^{20} bytes of byte-addressable main memory, and a cache of 32 blocks, where each cache block contains 16 bytes.

   a) How many blocks of main memory are there?
   b) What is the format of a memory address as seen by the cache, i.e., what are the sizes of the tag, block, and offset fields?
   c) To which cache block will the memory address 0x0DB63 map?

Ans.

a) $2^{20}/2^4 = 2^{16}$

b) 20-bit addresses with 11 bits in the tag field, 5 in the block field, and 4 in the offset field

c) 0x0DB63 = 00001100101 10110 0111, which implies Block 22 (or block 0x16)

---

2. Suppose a computer using direct mapped cache has 2^{32} byte of byte-addressable main memory, and a cache of 1024 blocks, where each cache block contains 32 bytes.
   a) How many blocks of main memory are there?
   b) What is the format of a memory address as seen by the cache, i.e., what are the sizes of the tag, block, and offset fields?
   c) To which cache block will the memory address 0x000063FA map?

Ans.

a) $2^{32}/2^5 = 2^{27}$

b) 32 bit addresses with 17 bits in the tag field, 10 in the block field, and 5 in the offset field

c) 000063FA = 00000000000000000 1100011111 11010, which implies Block 799

---

3. Suppose a computer using direct mapped cache has 2^{32} bytes of byte-addressable main memory and a cache size of 512 bytes, and each cache block contains 64 bytes.
   a) How many blocks of main memory are there?
   b) What is the format of a memory address as seen by cache, i.e., what are the sizes of the tag, block, and offset fields?
   c) To which cache block will the memory address 0x13A4498A map?

Ans.

a) $2^{32}/2^6 = 2^{26}$

b) 32 bit addresses with 23 bits in the tag field, 3 in the block field, and 6 in the offset field (cache has 512 bytes, so has $2^9/2^6 = 2^3$ blocks)

c) 12A4498A = 0001 0111 1010 0100 0100 1001 1000 1010, which implies Block 6.

♦4. Suppose a computer using fully associative cache has 2^{16} bytes of byte-addressable main memory and a cache of 64 blocks, where each cache block contains 32 bytes.

♦a) How many blocks of main memory are there?
♦b) What is the format of a memory address as seen by the cache, i.e., what are the sizes of the tag and offset fields?
♦c) To which cache block will the memory address 0xF8C9 map?

Ans.

a) $2^{16}/2^5 = 2^{11}$
b) 16 bit addresses with 11 bits in the tag field and 5 in the offset field
c) Because it is associative cache, it can map anywhere.

---

5. Suppose a computer using fully associative cache has 2^{24} bytes of byte-addressable main memory and a cache of 128 blocks, where each cache block contains 64 bytes.

a) How many blocks of main memory are there?
b) What is the format of a memory address as seen by the cache, i.e., what are the sizes of the tag and offset fields?
c) To which cache block will the memory address 0x01D872 map?

Ans.

a) $2^{24}/2^6 = 2^{18}$
b) 24 bit addresses with 18 bits in the tag field and 6 in the offset field
c) Since it's associative cache, it can map anywhere

---

6. Suppose a computer using fully associative cache has 2^{24} bytes of byte-addressable main memory and a cache of 128 blocks, where each block contains 64 bytes.

a) How many blocks of main memory are there?
b) What is the format of a memory address as seen by the cache, i.e., what are the sizes of the tag and offset fields?
c) To which cache block will the memory address 0x01D872 map?

Ans.

a) $2^{24}/2^6 = 2^{18}$
b) 24 bit addresses with 18 bits in the tag field and 6 in the offset field
c) Since it is associative cache, it can map anywhere

♦7. Assume a system's memory has 128M bytes. Blocks are 64 bytes in length and the cache consists of 32K blocks. Show the format for a main memory address assuming a 2-way set

associative cache mapping scheme and byte addressing. Be sure to include the fields as well as their sizes.

Ans.

Each address has 27 bits, and there are 7 in the tag field, 14 in the set field, and 6 in the offset field.

---

8. A 2-way set-associative cache consists of four sets. Main memory contains 2K blocks of eight bytes each and byte addressing is used.

   a) Show the main memory address format that allows us to map addresses from main memory to cache. Be sure to include the fields as well as their sizes.
   b) Compute the hit ratio for a program that loops 3 times from addresses 0x8 to 0x33 in main memory. You may leave the hit ratio in terms of a fraction.

Ans.

a) $2K * 2^3 = 2^{14}$, so we have 14-bit addresses with 9 bits in the tag field, 2 bits in the set field (since we have four sets), and 3 in the offset field

b) First iteration of the loop: Address 0x8 is a miss, then entire block brought into Set 1. 0x9 through 0xE are then hits. 0xF is a miss, the entire block is brought into Set 2, 0x11 through 0x17 are hits. 0x18 is a miss, the entire block is brought into Set 3, 0x19 through 0x1E are hits. Ox20 is a miss, the entire block is brought into Set 0, 0x21 through 0x27 are then hits. 0x28 is a miss, the entire block is brought into Set 1 (note we do NOT have to throw out the block with address 0x8 as this is 2-way set associative), 0x29 through 0x24 are hits. 0x30 is a miss, the entire block is brought into Set 2, 0x31 through 0x33 are hits. For the first iteration of the loop, we have 6 misses, and 5*7 + 3 hits, or 38 hits. On the remaining iterations, we have 5*8+4 hits, or 44 hits each, for 88 more hits. Therefore, we have 6 misses and 126 hits, for a hit ratio of 126/132, or 95.45%.

---

9. Suppose a byte-addressable computer using set associative cache has 2^{16} bytes of main memory and a cache of 32 blocks, and each cache block contains 8 bytes.

   a) If this cache is 2-way set associative, what is the format of a memory address as seen by the cache, that is, what are the sizes of the tag, set, and offset fields?
   b) If this cache is 4-way set associative, what is the format of a memory address as seen by the cache?

Ans.

a) A total of 2^{16} bytes of main memory implies we have 16 bits in an address. Cache contains 2^5 blocks, but each set must have 2 blocks, so we have $2^5/2=2^4$ sets. Therefore our 16-bit address is divided into 9 bits for the tag field, 4 bits for the set field, and 3 bits for the offset field.

b) The 32 blocks in cache must now be divided into sets with 4 blocks each, implying we have only 8 sets. The tag field would now have 10 bits, the set field 3 bits, and the offset field 3 bits.

---

10. Suppose a byte-addressable computer using set associative cache has 2^{21} byes of main memory and a cache of 64 blocks, where each cache block contains 4 bytes.

    a) If this cache is 2-way set associative, what is the format of a memory address as seen by the cache, that is, what are the sizes of the tag, set, and offset fields?
    b) If this cache is 4-way set associative, what is the format of a memory address as seen by the cache?

Ans.

a) 2^{21} bytes of main memory implies we have 21 bits in an address. Cache contains 2^6 blocks, but each set must have 2 blocks, so we have $2^6/2=2^5$ sets. Therefore our 21-bit address is divided into 14 bits for the tag field, 5 bits for the set field, and 2 bits for the offset field.
b) The 64 blocks in cache must now be divided into sets with 4 blocks each, implying we have only 16 sets. The tag field would now have 15 bits, the set field 4 bits, and the offset field 2 bits.

---

*11. Suppose we have a computer that uses a memory address word size of 8 bits. This computer has a 16-byte cache with 4 bytes per block. The computer accesses a number of memory locations throughout the course of running a program.

Suppose this computer uses direct-mapped cache. The format of a memory address as seen by the cache is shown below:

Tag 4 bits	Block 2 bits	Offset 2 bits

The system accesses memory addresses in this exact order: 0x6E, 0xB9, 0x17, 0xE0, 0x4E, 0x4F, 0x50, 0x91, 0xA8, 0xA9, 0xAB, 0xAD, 0x93, and 0x94. The memory addresses of the first four accesses have been loaded into the cache blocks as shown below. (The contents of the tag are shown in binary and the cache "contents" are simply the address stored at that cache location.)

	Tag Contents	Cache Contents (represented by address)
Block 0	1110	0xE0
		0xE1
		0xE2
		0xE3

	Tag Contents	Cache Contents (represented by address)
Block 1	0001	0x14
		0x15
		0x16
		0x17

Block 2	1011	0xB8
		0xB9
		0xBA
		0xBB

Block 3	0110	0x6C
		0x6D
		0x6E
		0x6F

a) What is the hit ratio for the entire memory reference sequence given above, assuming we count the first four accesses as misses?
b) What memory blocks will be in the cache after the last address has been accessed?

Ans.

a)

Address	Hit or Miss
0x6E	Miss, brought into Block 3 with tag 0110 (as shown)
0xB9	Miss, brought into Block 2 with tag 1011 (as shown)
0x17	Miss. brought into Block 1 with tag 0001 (as shown)
0xE0	Miss, brought into Block 0 with tag 1110 (as shown)
0x4E	Miss, brought into Block 3 with tag 0100
0x4F	Hit
0x50	Miss, brought into Block 0 with tag 0101
0x91	Miss, brought into Block 0 with tag 1001
0xA8	Miss, brought into Block 2 with tag 1010
0xA9	Hit
0xAB	Hit
0xAD	Miss, brought into Block 3 with tag 1010
0x93	Hit
0x94	Miss, brought into Block 1 with tab 1001

Hit ratio is 4 hits out of 14 total accesses, or 28.6% (this is assuming we count the first 4 accesses as misses)

b) Block 0, with tag 1001, contains 0x90, 0x91, 0x92, 0x93
Block 1, with tag 1001, contains 0x94, 0x95, 0x96, 0x97
Block 2, with tag 1010, contains 0xA8, 0xA9, 0xAA, 0xAB
Block 3, with tag 1010, contains 0xAC, 0xAD, 0xAE, 0xAF

12. Given a byte-addressable memory with 256 bytes, suppose a memory dump yields the results shown below. The address of each memory cell is determined by its row and column. For example, memory address 0x97 is in the 9th row, 7th column, and contains the hexadecimal value 43. Memory location 0xA3 contains the hexadecimal value 58.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	**DC**	**D5**	**9C**	**77**	**C1**	**99**	**90**	**AC**	**33**	**D1**	**37**	**74**	**B5**	**82**	**38**	**E0**
1	**49**	**E2**	**23**	**FD**	**D0**	**A6**	**98**	**BB**	**DE**	**9A**	**9E**	**EB**	**04**	**AA**	**86**	**E5**
2	**3A**	**14**	**F3**	**59**	**5C**	**41**	**B2**	**6D**	**18**	**3C**	**9D**	**1F**	**2F**	**78**	**44**	**1E**
3	**4E**	**B7**	**29**	**E7**	**87**	**3D**	**B8**	**E1**	**EF**	**C5**	**CE**	**BF**	**93**	**CB**	**39**	**7F**
4	**6B**	**69**	**02**	**56**	**7E**	**DA**	**2A**	**76**	**89**	**20**	**85**	**88**	**72**	**92**	**E9**	**5B**
5	**B9**	**16**	**A8**	**FA**	**AE**	**68**	**21**	**25**	**34**	**24**	**B6**	**48**	**17**	**83**	**75**	**0A**
6	**40**	**2B**	**C4**	**1D**	**08**	**03**	**0E**	**0B**	**B4**	**C2**	**53**	**FB**	**E3**	**8C**	**0C**	**9B**
7	**31**	**AF**	**30**	**9F**	**A4**	**FE**	**09**	**60**	**4F**	**D7**	**D9**	**97**	**2E**	**6C**	**94**	**BC**
8	**CD**	**80**	**64**	**B3**	**8D**	**81**	**A7**	**DB**	**F1**	**BA**	**66**	**BE**	**11**	**1A**	**A1**	**D2**
9	**61**	**28**	**5D**	**D4**	**4A**	**10**	**A2**	**43**	**CC**	**07**	**7D**	**5A**	**C0**	**D3**	**CF**	**67**
A	**52**	**57**	**A3**	**58**	**55**	**0F**	**E8**	**F6**	**91**	**F0**	**C3**	**19**	**F9**	**BD**	**8B**	**47**
B	**26**	**51**	**1C**	**C6**	**3B**	**ED**	**7B**	**EE**	**95**	**12**	**7C**	**DF**	**B1**	**4D**	**EC**	**42**
C	**22**	**0D**	**F5**	**2C**	**62**	**B0**	**5E**	**DD**	**8E**	**96**	**A0**	**C8**	**27**	**3E**	**EA**	**01**
D	**50**	**35**	**A9**	**4C**	**6A**	**00**	**8A**	**D6**	**5F**	**7A**	**FF**	**71**	**13**	**F4**	**F8**	**46**
E	**1B**	**4B**	**70**	**84**	**6E**	**F7**	**63**	**3F**	**CA**	**45**	**65**	**73**	**79**	**C9**	**FC**	**A5**
F	**AB**	**E6**	**2D**	**54**	**E4**	**8F**	**36**	**6F**	**C7**	**05**	**D8**	**F2**	**AD**	**15**	**32**	**06**

The system from which this memory dump was produced contains 4 blocks of cache, where each block consists of eight bytes. Assume the following sequence of memory addresses take place: 0x2C, 0x6D, 0x86, 0x29, 0xA5, 0x82, 0xA7, 0x68, 0x80, and 0x2B.

a) How many blocks of main memory are there?
b) Assuming a direct mapped cache:
   i) Show the format for a main memory address assuming the system uses direct mapped cache. Specify field names and sizes.

ii) What does cache look like after the ten memory accesses have taken place? Draw the cache and show contents and tags.
iii) What is the hit rate for this cache on the given sequence of memory accesses?

c) Assuming a fully associative cache:
i) Show the format for a main memory address. Specify field names and sizes.
ii) Assuming all cache blocks are initially empty, blocks are loaded into the first available empty cache location, and cache uses a first-in-first-out replacement policy, what does cache look like after the ten memory accesses have taken place?
iii) What is the hit rate for this cache on the given sequences of memory accesses?

d) Assuming a 2-way set associative cache:
i) Show the format for a main memory address. Specify field names and sizes.
ii) What does cache look like after the ten memory accesses have taken place?
iii) What is the hit ratio for this cache on the given sequences of memory accesses?
iv) If a cache hit retrieves a value in 5ns, and retrieving a value from main memory requires 25ns, what is the average effective access time for this cache, assuming that all memory accesses exhibit the same hit rate as the sequence of 10 given, and assuming the system uses a non-overlapped (sequential) access strategy?

Ans.

a) 32

b) i) A total of 8 bits, with a 3-bit tag field, a 2-bit block, and 3-bit offset field.
ii) Block 0 of cache holds the memory block containing the address 80 (the values CD, 80, 64, B3, 8D, 81, A7, and DB), with a tag value of 100. Block 1 of cache holds the memory block containing the address 2B (the values 18, 3C, 9D, 1F, 2F, 7B, 44, and 1E), with a tag value of 001.
iii) The hit ratio for this cache is 10 misses out of 10 accesses, or 0%.

c) i) A total of 8 bits, with a 5-bit tag field and a 3-bit offset field.
ii) Block 0 contains the memory block for address 2C, with tag 00101; block 1 contains the memory block for address 6D, with tag 01101; block 2 contains the memory block for address 86, with tag 100000; and block 3 contains the memory block for address A5, with tag 10100.
iii) The hit rate is 6 hits out of 10 accesses, for a hit ratio of 60%.

d) i) A total of 8 bits, with a 4-bit tag field, a 1-bit set field, and a 3-bit offset field.
ii) Set 0 contains two blocks, one with the memory block for address 86 and tag 1000, and one with the memory block for address A5 and tag 1010. Set 1 contains two blocks, one with the memory block for 2C and tag 0010, and one with the memory block for address 6D and tag 0110.
iii) The hit rate is 6 hits out of 10 accesses, for a hit ratio of 60%.
iv) EAT = .6(5ns) + .4(5ns + 25ns) = 15ns

---

13. A direct-mapped cache consists of eight blocks. A byte-addressable main memory contains 4K blocks of eight bytes each. Access time for the cache is 22 ns and the time required to fill

a cache slot from main memory is 300 ns (this time will allow us to determine the block is missing and bring it into cache). Assume a request is always started in parallel to both cache and to main memory (so if it is not found in cache, we do not have to add this cache search time to the memory access). If a block is missing from cache, the entire block is brought into the cache and the access is restarted. Initially, the cache is empty.

a) Show the main memory address format that allows us to map addresses from main memory to cache. Be sure to include the fields as well as their sizes.
b) Compute the hit ratio for a program that loops 4 times from locations 0 to 67_{10} in memory.
c) Compute the effective access time for this program.

Ans.

a) 4K = 2^{12} blocks * 2^3 bytes each implies 2^{15} bytes of main memory, so we have 15 bits in an address. Cache contains 2^3 blocks, so the block field requires 3 bits. Each block has 2^3 bytes, so the offset field requires 3 bits. That leaves 9 bits for the tag field. Therefore our 15-bit address is divided into 9 bits for the tag field, 3 bits for the block field, and 3 bits for the offset field.

b) Block 0 of main memory (addresses 0 - 7) and Block 8 of main memory (addresses 64 - 67) must share a cache block. The remaining blocks are brought in and are not replaced. So for each access to Block 0, there is one miss and 7 hits. For each access to Block 8, there is one miss and 3 hits. The remaining blocks have one miss each, with all other accesses being hits. If we loop 4 times, we have:

Block 0: 4 misses, 28 hits

Block 1: 1 miss, 31 hits

Block 2: 1 miss, 31 hits

Block 3: 1 miss, 31 hits

Block 4: 1 miss, 31 hits

Block 5: 1 miss, 31 hits

Block 6: 1 miss, 31 hits

Block 7: 1 miss, 31 hits

Block 8: 4 misses, 12 hits for a total of 15 misses, 257 hits, or a hit ratio of 94.49%.

c) The effective access time is .9449(22ns) + .0551(300ns + 22ns)

---

14. Consider a byte-addressable computer with 24-bit addresses, a cache capable of storing a total of 64K bytes of data and blocks of 32 bytes. Show the format of a 24-bit memory address for:

a) direct mapped
b) associative
c) 4-way set associative

Ans.

a) 64K = $2^6 2^{10} = 2^{16}$; $2^{16}/2^5 = 2^{11}$ blocks in cache, so 11 bits are needed for the block field. 5 bits are needed for the offset field, leaving 8 for the tag.
b) Again, 5 bits are needed for the offset field, leaving 19 for the tag.

c) There are $2^{11}/2^2 = 2^9$ sets in cache, so 9 bits are needed for the set field. We still need 5 bits for the offset field, leaving 10 for the tag field.

---

*15. Suppose a byte-addressable computer using 4-way set associative cache has 2^{16} words of main memory (where each word is 32 bits) and a cache of 32 blocks, where each block is 4 words. Show the main memory address format for this machine. (Hint: Because this architecture is byte-addressable, and the number of addresses is critical in determining the address format, you must convert everything to bytes.)

Ans.

If main memory is 2^{16} words, then it is $2^{16}x2^2=2^{18}$ bytes, so there are 18 bits in a main memory address. Each block is 4 words, or $2^2x2^2=2^4$ bytes. Therefore, the offset field must have 4 bits. There are $2^5/2^2=2^3$ sets, so the set field has 3 bits. That leaves 11 bits for the tag.

---

16. Assume a direct-mapped cache that holds 4096 bytes, where each block is 16 bytes. Assuming an address is 32 bits and that cache is initially empty, complete the table below. (You should use hexadecimal numbers for all answers.) Which, if any of the addresses will cause a collision (forcing the block that was just brought in to be overwritten) if they are accessed one right after the other?

Address	TAG	Cache location (block)	Offset within block
0x0FF0FABA			
0x00000011			
0x0FFFFFFE			
0x23456719			
0xCAFEBABE			

Ans.

Address	TAG	Cache location (block)	Offset within block
0x0FF0FABA	0x0FF0F	0xAB	0xA
0x00000011	0x00000	0x01	0x1
0x0FFFFFFE	0x0FFFF	0xFF	0xE
0x23456719	0x23456	0x71	0x9
0xCAFEBABE	0xCAFEB	0xAB	0xE

The first and last addresses both map to cache block 0xAB and therefore will cause a collision.

---

17. Redo Problem #16, assuming now that cache is 16-way set associative.

Address	TAG	Cache location (set)	Offset within block
0x0FF0FABA			
0x00000011			
0x0FFFFFFE			
0x23456719			
0xCAFEBABE			

Ans.

Address	TAG	Cache location (set)	Offset within block
0x0FF0FABA	0x0FF0FA	0xB	0xA
0x00000011	0x000000	0x1	0x1
0x0FFFFFFE	0x0FFFFF	0xF	0xE
0x23456719	0x234567	0x1	0x9
0xCAFEBABE	0xCAFEBA	0xB	0xE

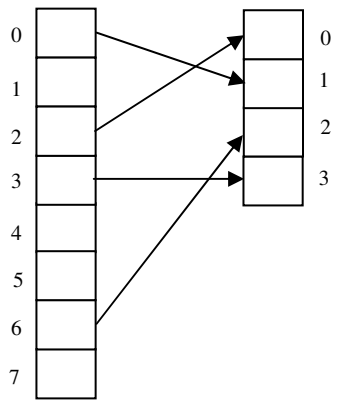
The first and last addresses both map to cache set 0xAB; the 2nd and 4th addresses both map to set 0x1. However, since there are multiple blocks in each set, these should not cause a collision. (Since cache was initially empty we don't have to worry about the set being full.)

---

18. Suppose a process page table contains the entries shown below. Using the format shown in Figure 6.17a, indicate where the process pages are located in memory.

Frame	Valid Bit
1	1
--	0
0	1
3	1
--	0
--	0
2	1
--	0

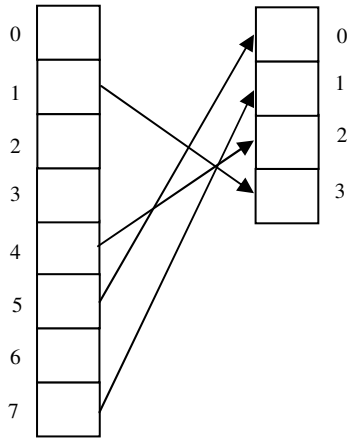
Ans.



♦19. Suppose a process page table contains the entries shown below. Using the format shown in Figure 6.22a, indicate where the process pages are located in memory.

Frame	Valid Bit
--	0
3	1
--	0
--	0
2	1
0	1
--	0
1	1

Ans.



---

20. Suppose you have a byte-addressable virtual address memory system with 8 virtual pages of 64 bytes each, and 4 page frames. Assuming the following page table, answer the questions below:

Page #	Frame #	Valid Bit
0	1	1
1	3	0
2	-	0
3	0	1
4	2	1
5	-	0
6	-	0
7	-	0

a) How many bits are in a virtual address?

b) How many bits are in a physical address?

c) What physical address corresponds to the following virtual addresses (if the address causes a page fault, simply indicate this is the case)?

i) 0x00
ii) 0x44
iii) 0xC2
iv) 0x80

Ans.

a) 8 virtual pages of size 64 bytes each is $2^3 x 2^6 = 2^9$. Therefore, each virtual address has 9 bits.
b) There are 4 pages frames of size 64 each, or $2^2 x 2^6 = 2^8$. So each physical address has 8 bits.
c)
   i) 0x0 = 000 000000 so this address is on page 0, offset 0. Page 0 maps to frame 1. Substituting 01 for 000, we get 01 000000, or 0x10.
   ii) 0x44 = 001 000100 so this address is on page 1, offset 4. Page 1 maps to frame 3. Substituting 11 for 001, we get 11 000100, or 0xC4.
   iii) 0xC2 = 011 000010 so this address is on page 3, offset 2. Page 3 maps to frame 0. Substituting 00 for 011, we get 00 000010, or 0x02.
   iv) 0x80 = 010 000000 so this address is on page 2, offset 0. Page 2 is not currently in memory so this generates a page fault.

---

21. Suppose we have 2^{10} bytes of virtual memory and 2^8 bytes of physical main memory. Suppose the page size is 2^4 bytes.

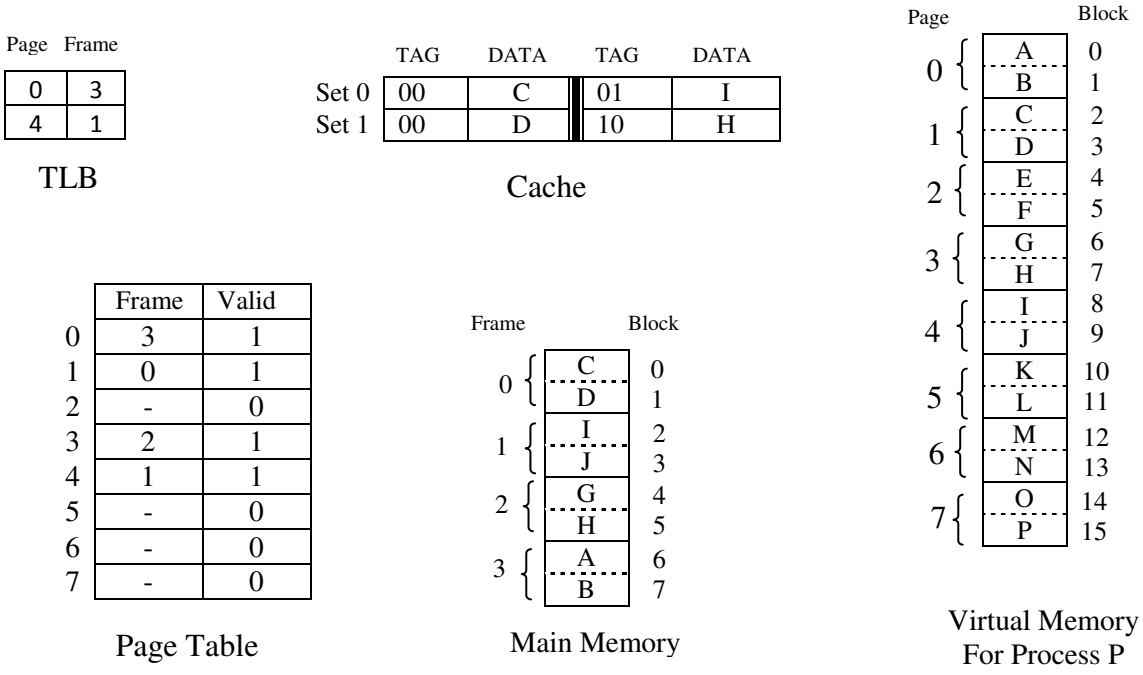
a) How many pages are there in virtual memory?
b) How many page frames are there in main memory?
c) How many entries are in the page table for a process that uses all of virtual memory?

Ans.

a) $2^{10}/2^4 = 2^6$, so there are 64 pages in virtual memory
b) $2^8/2^4 = 2^4$, so there are 16 page frames in virtual memory
c) The page table must have an entry for each virtual page, so it must have 64 entries.

---

*22. You have a byte-addressable virtual memory system with a two-entry TLB, a 2-way set associative cache and a page table for a process P. Assume cache blocks of 8 bytes and page size of 16 bytes. In the system below, main memory is divided up into blocks, where each block is represented by a letter. Two blocks equals one frame.



Given the system state as depicted above, answer the following questions:

a) How many bits are in a virtual address for process P? Explain.
b) How many bits are in a physical address? Explain.
c) Show the address format for virtual address 0x12 (specify field name and size) that would be used by the system to translate to a physical address and then translate this virtual address into the corresponding physical address. (Hint: convert the address to its binary equivalent and divide it into the appropriate fields.) Explain how these fields are used to translate to the corresponding physical address.
d) Given virtual address 0x06 converts to physical address 0x36. Show the format for a physical address (specify the field names and sizes) that is used to determine the cache location for this address. Explain how to use this format to determine where physical address 0x36 would be located in cache. (Hint: convert 0x36 to binary and divide it into the appropriate fields.)
e) Given virtual address 0x19 is located on virtual page 1, offset 9. Indicate exactly how this address would be translated to its corresponding physical address and how the data would be accessed. Include in your explanation how the TLB, page table, cache and memory are used.

Ans.

a) $2^3 * 2^4 = 2^7$, so there are 7 bits in a virtual address
b) $2^2 * 2^4 = 2^6$, so there are 6 bits in a virtual address

c) 0x12 = 001 0010 (where 001 is the page field and 0010 is the offset). Using the page table, and going to entry 1, we see that page 1 maps to frame 0, so the actual physical address would be 00 0010, or 2.

d) 0x36 = 11 0 110, where 11 is the tag, 0 is the set, and 110 is the offset. Therefore, this would map to Set 0 in cache. Once there, if the tag is found, the block is in cache. If not, it is a miss.

e) 0x19 = 001 1001, where 001 is the virtual page, and 1001 is the offset. (This maps to physical page 00 with offset 1001.) The TLB would first be checked to see if the pair (1,0) was present (virtual page 1, physical frame 0). If so, 00 can be substituted for 001, giving the actual physical address 00 1001. If the entry is not found in the TLB, the page table must be accessed, at which time the physical address 00 1001 is determined. At this point (regardless of whether we found the physical address using the TLB or the page table), the cache is checked to see if the block containing this physical address is currently cached. The memory address 001001 is divided up into 00 1 001, where 00 is the tag, 1 is the set, and 001 is the offset. Set 1 in cache is then checked for the tag 00 (it would be found as address 0x19 is in block D). The value in cache is used. (If it had not been found in cache, main memory would be accessed.)

---

23. Given a virtual memory system with a TLB, a cache, and a page table. Assume the following:

- A TLB hit requires 5ns
- A cache hit requires 12ns
- A memory reference requires 25ns
- A disk reference requires 200ms (this includes updating the page table, cache, and TLB)
- The TLB hit ratio is 90%
- The cache hit rate is 98%
- The page fault rate is .001%
- On a TLB or cache miss, the time required for access includes a TLB and/or cache update, but the access is *not* restarted
- On a page fault, the page is fetched from disk, all updates are performed but the access *is* restarted
- All references are sequential (no overlap, nothing done in parallel)

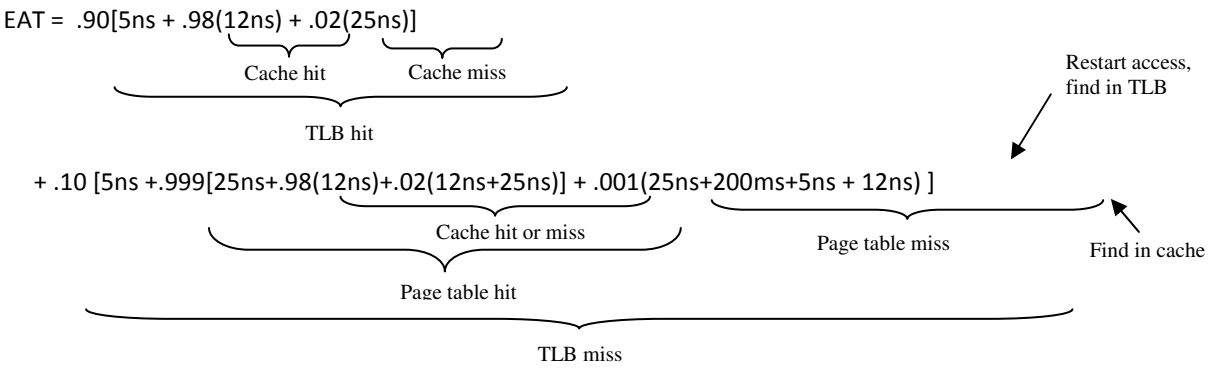
For each of the following, indicate whether or not it is possible. If so, specify the time required for accessing the requested data.

a) TLB hit, cache hit
b) TLB miss, page table hit, cache hit
c) TLB miss, page table hit, cache miss
d) TLB miss, page table miss, cache hit
e) TLB miss, page table miss

Write down the equation to calculate the effective access time.

Ans.

a) Possible, 5ns (TLB access) + 12ns (cache access)
b) Possible, 5ns (TLB access) + 25ns (page table reference) + 12ns (cache access)
c) Possible, 5ns (TLB access) + 25ns (page table reference) + 12ns (cache access) + 25ns (main memory reference)
d) Not possible (the cache value would be stale if the page no longer resides in page table)
e) Possible, 5ns (TLB access) + 25ns (page table reference) + 200ms (disk reference) + 5nx (TLB, since access is restarted) + 12ns (cache hit)



---

24. Does a TLB miss always indicate that a page is missing from memory? Explain.

Ans.

No. While the page could be missing from memory, a TLB miss simply means that page is not cached in the TLB.

---

25. A system implements a paged virtual address space for each process using a one-level page table. The maximum size of virtual address space is 16MB. The page table for the running process includes the following valid entries (the → notation indicates that a virtual page maps to the given page frame, that is, it is located in that frame):

Virtual page 2 → page frame 4	Virtual page 4 → page frame 9
Virtual page 1 → page frame 2	Virtual page 3 → page frame 16
Virtual page 0 → page frame 1	

The page size is 1024 bytes and the maximum physical memory size of the machine is 2MB.

a) How many bits are required for each virtual address?
b) How many bits are required for each physical address?

c) What is the maximum number of entries in a page table?
d) To which physical address will the virtual address 0x5F4 translate?
e) Which virtual address will translate to physical address 0x400?

Ans.

a) There are 16MB, or $2^4 x 2^{20}$ addresses, so we need 24 bits for a virtual address.
b) Main memory is 2MB, or $2^1 x 2^{20}$, so we need 21 bits for a physical address.
c) There are $2^{24}/2^{10}$ pages in virtual memory, so the page table can have 2^{14} entries.
d) Address 0x5F4 is on page 1 (page 0 contains addresses 0x000 – 0x3FF; page 1 contains 0x400 – 0x7FF), located at offset 0x1F4. Page 1 maps to frame 2 (which begins with address 0x800), so virtual address 0x5F4 maps to physical address 0x800+0x1F4= 0x9F4. You can also find the binary equivalent of 0x5F4 (000000000000010111110100) and divide it into two pieces: the first 14 bits are the page, and the last 10 are the offset. Replace the first 14 bits (00000000000001) by (00000000000010), to get the physical address 000000000000100111110100, or 0x009F4.
e) Physical address 0x400 is at offset 0 in frame 1. Virtual page 0 maps to frame 1, so this is virtual address 0.

---

26. a) If you are a computer builder trying to make your system as price-competitive as possible, what features and organization would you select for its memory hierarchy?

b) If you are a computer buyer trying to get the best performance from a system, what features would you look for in its memory hierarchy?

Ans.

a) To be competitive, the system would need cache, but it could be the cheaper, non-associative cache (direct-mapped). To keep the price low, you light leave off level 2 cache entirely. There could also be a very small TLB. Cheaper main memory could be used, but putting a large hard drive would interest buyers.
b) You should look for a machine with two-level cache (level one being some type of associative memory). Note that a larger cache isn't necessarily better. Up to a certain point, more cache can help raise the hit ratio, but after a point, the hit ratio doesn't change much (so don't pay for what you don't need). A good-sized TLB will help with performance as well. A large, fast main memory is a definite plus.

---

*27. Consider a system that has multiple processors where each processor has its own cache, but main memory is shared among all processors.

a) Which cache write policy would you use?
b) **The Cache Coherency Problem.** With regard to the system just described, what problems are caused if a processor has a copy of memory block *A* in its cache and a second processor, also having a copy of *A* in its cache, then updates main memory block

A? Can you think of a way (perhaps more than one) of preventing this situation, or lessening its effects?

Ans.

a) Write through should be used to maintain consistency. If some processors have a value cached, and one processor changes that value, the other processors would not know about that value. With a write through cache (in addition to a broadcast "invalidate" message) the processors would not be using stale values.

b) As mentioned in the answer for part a, the problem is stale data. One way to solve this problem is to invalidate stale entries. A way to prevent the situation would be to require processors to specify whether the values were for writing or reading. Shared reading would be ok; however, when a processor wanted to write, it would have to either wait until the readers were done, or send an invalidate message to those readers. Exclusive access by writers would then be allowed.

---

*28. Pick a specific architecture (other than the one covered in this chapter). Do research to find out how your architecture approaches the concepts introduced in this chapter, as was done for Intel's Pentium.

Ans.

No answer given.

---

29. Name two ways that, as a programmer, you can improve cache performance.

Ans.

Programmers should focus on improving the reference locality. This can be done by using cache-conscious algorithms (for example, change a program's data access pattern to optimize locality in nested loops used in matrices by interchanging loops) or a program's data organization and layout (such as using cache-conscious data structures).

---

30. Look up a specific vendor's specifications for memory and report the memory access time, cache access time, and cache hit rate (and any other data the vendor provides).

Ans.

No answer given.
