## 7.11 Exercises

**7.1** [10] <§7.2> Describe the general characteristics of a program that would exhibit very little temporal and spatial locality with regard to data accesses. Provide an example program (pseudocode is fine).

**7.2** [10] <§7.2> Describe the general characteristics of a program that would exhibit very high amounts of temporal locality but very little spatial locality with regard to data accesses. Provide an example program (pseudocode is fine).

**7.3** [10] <§7.2> Describe the general characteristics of a program that would exhibit very little temporal locality but very high amounts of spatial locality with regard to data accesses. Provide an example program (pseudocode is fine).

**7.4** [10] <§7.2> Describe the general characteristics of a program that would exhibit very little temporal and spatial locality with regard to instruction fetches. Provide an example program (pseudocode is fine).

**7.5** [10] <§7.2> Describe the general characteristics of a program that would exhibit very high amounts of temporal locality but very little spatial locality with regard to instruction fetches. Provide an example program (pseudocode is fine).

**7.6** [10] <§7.2> Describe the general characteristics of a program that would exhibit very little temporal locality but very high amounts of spatial locality with regard to instruction fetches. Provide an example program (pseudocode is fine).

**7.7** [10] <§7.2> Here is a series of address references given as word addresses: 1, 4, 8, 5, 20, 17, 19, 56, 9, 11, 4, 43, 5, 6, 9, 17. Assuming a direct-mapped cache with 16 one-word blocks that is initially empty, label each reference in the list as a hit or a miss and show the final contents of the cache.

**7.8** [10] <§7.2> Using the series of references given in Exercise 7.7, show the hits and misses and final cache contents for a direct-mapped cache with four-word blocks and a *total size* of 16 words.

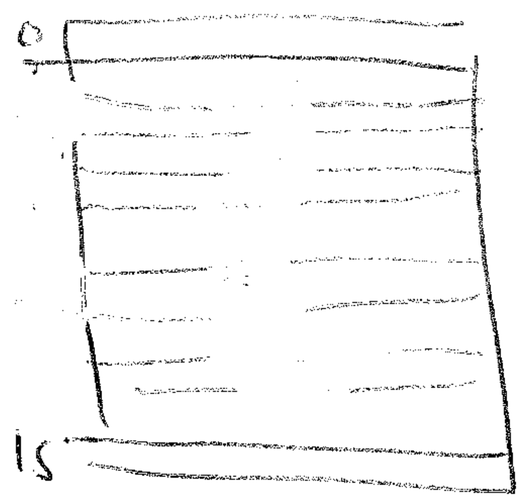
**7.9** [10] <§7.2> Compute the total number of bits required to implement the cache in Figure 7.10 on page 557. This number is different from the size of the cache, which usually refers to the number of bytes of data stored in the cache. The number of bits needed to implement the cache represents the total amount of memory needed for storing all of the data, tags, and valid bits.

7-7 set of references:

1, 4, 8, 5, 20, 17, 19, 56, 9, 11, 4, 43, 5, 6, 9, 17

Given no word addresses.

Assume direct mapped cache with 16 ~~word~~ blocks (each 1 word). Find hit, miss




mapping = Address module # of blocks

| Address | Map | hit/miss |
|---|---|---|
| 1 | 1 | miss |
| 4 | 4 | miss |
| 8 | 8 | miss |
| 5 | 5 | miss |
| 20 | 4 | miss |
| 17 | 1 | miss |
| 19 | 3 | miss |
| 56 | 8 | miss |
| 9 | 9 | miss |
| 11 | 11 | miss |
| 4 | 4 | miss (20) |
| 43 | 11 | miss (11) |
| 5 | 5 | hit |
| 6 | 6 | miss |
| 9 | 9 | hit |
| 17 | 1 | hit |

3 hit / 16

7-8 same reference if block size = 4 words and cache size = 16 words.

# blocks = 4

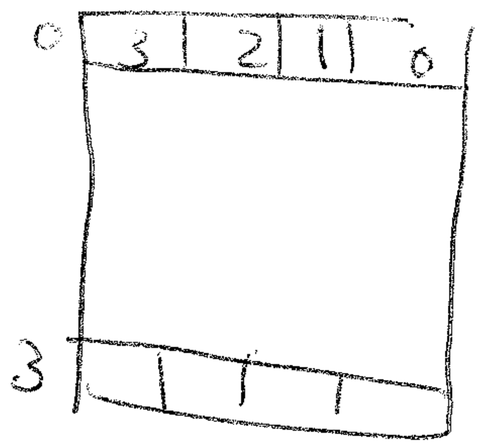
mapping

memory block number = Address / 4

mapped memory block # modulo 4

| Address | Memory block # | Cache block | hit/miss |
|---|---|---|---|
| 1 | 0 | 0 | miss |
| 4 | 1 | 1 | miss |
| 8 | 2 | 2 | miss |
| 5 | 1 | 1 | hit — |
| 20 | 5 | 0 | miss |
| 17 | 4 | 1 | miss (5) |
| 19 | 4 | 1 | hit (4) — |
| 56 | 14 | 2 | miss (2) |
| 9 | 2 | 2 | miss (14) |
| 11 | 2 | 2 | hit (2) — |
| 4 | 1 | 1 | miss (20) |
| 43 | 10 | 2 | miss (2) |
| 5 | 1 | 1 | hit — |
| 6 | 1 | 1 | hit — |
| 9 | 2 | 2 | miss (10) |
| 17 | 4 | 1 | hit (4) — |

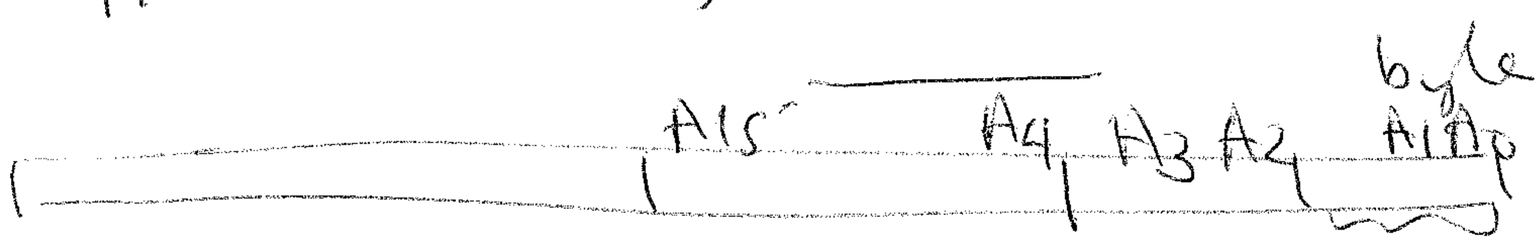
blk # Cache




6 hits

7-9 Find cache size (data, tag, valid bit)

has 4K entries, block = four words = 16 byte.




byte select = $\log_2$ 4 byte = 2 bits

Word select in block = $\log_2$ 4 = 2 bits

index = $\log_2$ 4K = 12 bits

Tag = 32 − 12 − 2 − 2 = 16 bits + 1 bit valid

= 17 bits X 4K = 2 byte X 4K + 4K bit

Data = 16B X 4K = 64KB

Total = 72 KB + 4K bit valid

**7.10** [10] <§7.2> Find a method to eliminate the AND gate on the valid bit in Figure 7.7 on page 549. (Hint: You need to change the comparison.)

**7.11** [10] <§7.2> Consider a memory hierarchy using one of the three organizations for main memory shown in Figure 7.13 on page 561. Assume that the cache block size is 16 words, that the width of organization b of the figure is four words, and that the number of banks in organization c is four. If the main memory latency for a new access is 10 cycles and the transfer time is 1 cycle, what are the miss penalties for each of these organizations?

**7.12** [10] <§7.2> {Ex. 7.11} Suppose a processor with a 16-word block size has an effective miss rate per instruction of 0.5%. Assume that the CPI without cache misses is 1.2. Using the memories described in Figure 7.13 on page 561 and Exercise 7.11, how much faster is this processor when using the wide memory than when using narrow or interleaved memories?

**7.13** [15] <§7.2> Cache C1 is direct-mapped with 16 one-word blocks. Cache C2 is direct-mapped with 4 four-word blocks. Assume that the miss penalty for C1 is 8 clock cycles and the miss penalty for C2 is 11 clock cycles. Assuming that the caches are initially empty, find a reference string for which C2 has a lower miss rate but spends more cycles on cache misses than C1. Use word addresses.

**7.14** [15] <§7.2> For the caches in Exercise 7.13, find a series of references for which C2 has more misses than C1. Use word addresses.

In More Depth

## Average Memory Access Time

To capture the fact that the time to access data for both hits and misses affects performance, designers often use average memory access time (AMAT) as a way to examine alternative cache designs. Average memory access time is the average time to access memory considering both hits and misses and the frequency of different accesses; it is equal to the following:

$$\text{AMAT} = \text{Time for a hit} + \text{Miss rate} \times \text{Miss penalty}$$

AMAT is useful as a figure of merit for different cache systems.

**7.15** [5] <§7.2> Find the AMAT for a machine with a 2-ns clock, a miss penalty of 20 clock cycles, a miss rate of 0.05 misses per instruction, and a cache access time (including hit detection) of 1 clock cycle. Assume that the read and write miss penalties are the same and ignore other write stalls.

**7.16** [5] <§7.2> {Ex. 7.15} Suppose we can improve the miss rate [illegible] misses per reference by doubling the cache size. This causes the cache access time to increase to 1.2 clock cycles. Using the AMAT as a metric, determine if this is a good trade-off.

**7.17** [10] <§7.2> [Ex. 7.16} If the cache access time determines the processor's clock cycle time, which is often the case, AMAT may not correctly indicate whether one cache organization is better than another. If a machine's clock cycle time must be changed to match that of a cache, is it a good trade-off? Assume the machines are identical except for the cache miss rate and the number of cache miss cycles; assume 1.5 references per instruction and a CPI without cache misses of 2. The miss penalty is 20 cycles for both machines.

**7.18** [10] <§§7.2, B.5> You have been given 18 32K × 8-bit SRAMs to build an instruction cache for a processor with a 32-bit address. What is the largest cache (i.e., the largest size of the data storage area in bytes) direct-mapped instruction cache that you can build with one-word (32-bit) blocks? Show the breakdown of the address into its cache access components (for an example see Figure 7.8) and describe how the various SRAM chips will be used. (Hint: You may not need all of them.)

**7.19** [10] <§§7.2, B.5> This exercise is similar to Exercise 7.18, except that this time you decide to build a direct-mapped cache with four-word blocks using Figure 7.10. Once again show the breakdown of the address and describe how the chips are used.

**7.20** [10] <§7.3> Using the series of references given in Exercise 7.7, show the hits and misses and final cache contents for a two-way set-associative cache with one-word blocks and a *total size* of 16 words. Assume LRU replacement.

**7.21** [10] <§7.3> Using the series of references given in Exercise 7.7, show the hits and misses and final cache contents for a fully associative cache with one-word blocks and a *total size* of 16 words. Assume LRU replacement.

**7.22** [10] <§7.3> Using the series of references given in Exercise 7.7, show the hits and misses and final cache contents for a fully associative cache with two-word blocks and a *total size* of 16 words. Assume LRU replacement.

**7.23** [5] <§7.3> Associativity usually improves the miss ratio, but not always. Give a short series of address references for which a two-way set-associative cache with LRU replacement would experience more misses than a direct-mapped cache of the same size.

**7.24** [15] <§7.3> Suppose a computer's address size is $k$ bits (using byte addressing), the cache size is $S$ bytes, the block size is $B$ bytes, and the cache is two-way set-associative. Assume that $B$ is a power of two, so $B = 2^b$. Figure [illegible]

7-15

2ns clock cycle, miss penalty at 20 cycles

miss rate = .05

find average Time / Instruction.

---

T = hit time + miss rate x miss penalty

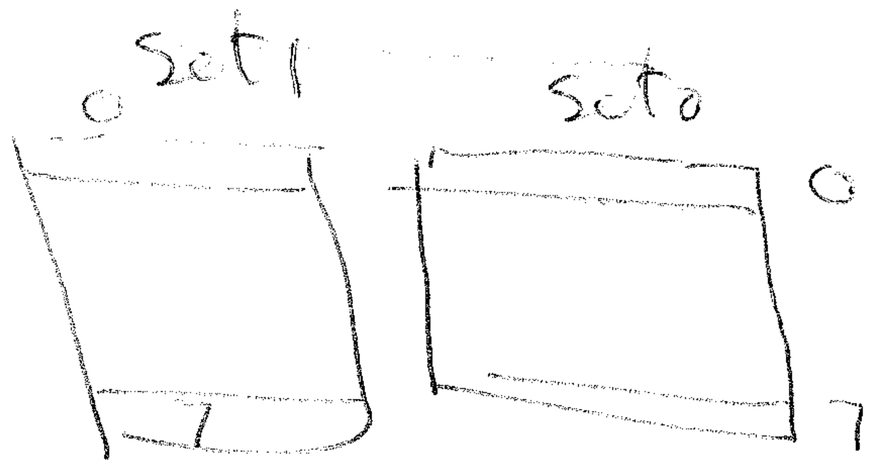
= 2ns + .05 x 20 x 2ns

= 2 + 2 = 4ns

---

7-20 use same sequence, assume two way set associative, LRU, size = 16 word, block = 1 word

| # Hits | cache set 0 hit/ms | cache set 1 hit/ms | |
|---|---|---|---|
| 1 | 1 | ms | |
| 4 | 4 | ms | |
| 8 | 4 | ms | E |
| 5 | 5 | ms | |
| 20 | | | 4 ms |
| 17 | | | 1 ms |
| 19 | 3 | ms | |
| 56 | | | 0 ms |
| 9 | 1 | ms | |
| 11 | | | 3 ms |
| 4 | 4 | hit | |
| 43 | 3 | ms | |
| 5 | 5 | hit | |
| 6 | 6 | ms | |
| 9 | 1 | hit | |
| 17 | 1 | | 1 hit |




4 hits

→ LRU

→ LRU

**7.30** [10] <§§7.3, B.5> This exercise is similar to Exercise 7.18, except that the time you decide to build a three-way set-associative cache with [illegible] blocks. Once again show the breakdown of the address (see Figure 7.18 for an example of a four-way set-associative cache) and describe how the data are used. Note that each SRAM will only perform a single read per cache access.

**7.31** [5] <§§7.2–7.4> Rank each of the possible event combinations appearing in the example on page 595 according to how frequently you think they would occur.

**7.32** [15] <§7.4> Consider a virtual memory system with the following properties:

- 40-bit virtual byte address
- 16-KB pages
- 36-bit physical byte address

What is the total size of the page table for each process on this machine, assuming that the valid, protection, dirty, and use bits take a total of 4 bits, and that all the virtual pages are in use? (Assume that disk addresses are not stored in the page table.)

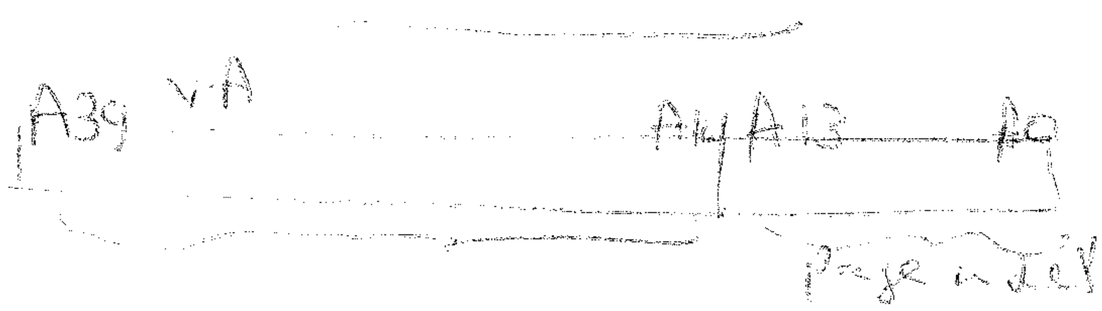
**7.33** [15] <§7.4> Assume that the virtual memory system of Exercise 7.32 is implemented with a two-way set-associative TLB with a total of 256 TLB entries. Show the virtual-to-physical mapping with a figure like Figure 7.25 on page 593. Make sure to label the width of all fields and signals.

**7.34** [15] <§7.3> Assume that the cache for the system described in Exercise 7.32 is two-way set associative and has eight-word blocks and a total size of 16 KB. Show the cache organization and access using the same format as Figure 7.19 on page 574.

**7.35** [15] <§7.4> Page tables require fairly large amounts of memory (as described in the elaboration on page 587), even if most of the entries are not valid. One solution is to use a hierarchy of page tables. The virtual page number, as described in Figure 7.21 on page 582, can be broken up into two pieces, a "page table number" and a "page table offset." The page table number can be used to index a first-level page table that provides a physical address for a second-level page table, assuming it resides in memory (if not, a first-level page fault will occur and the page table itself will need to be brought in from disk). The page table offset is used to index into the second-level page table to retrieve the physical page number. One obvious way to arrange such a scheme is to have the second-level page tables occupy exactly one page of memory. Assuming a 32-bit virtual address space with 4-KB pages and 4 bytes per page table entry, how many bytes will each program need to use to store the [illegible]

1-32 Assume V/M with:
- 40 bit VA
- 16KB pages
= 36 bit physical Address

Find size of 1 page Table using 4 bits for Protection



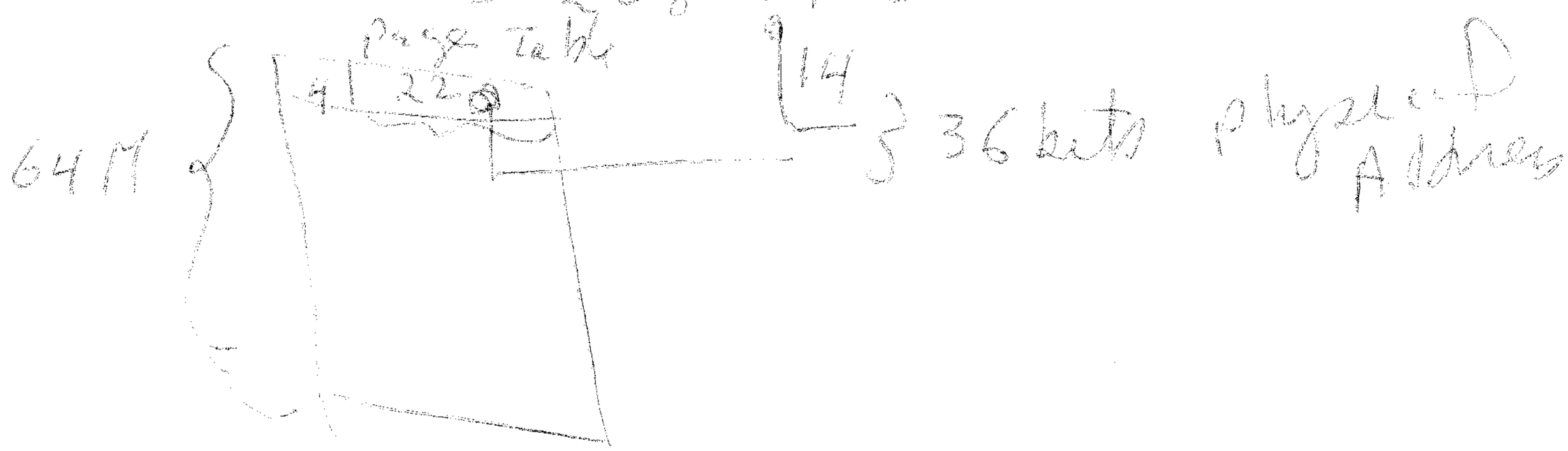

Page index = $\log_2$ 16KB = 14 bits

- size of Virpage number in bits = 40 - 14 = 26 bits
- number of entries in page Table
$= 2^{26} = 64$ Million
- Each entry has = a physical page number
+ 4 bits

physical page number = 36 - 14 = 22 bits

Each entry has 22 + 4 = 26 bits

Total size of page Table = 64 M X 26 bits
= 208 MB



7-33. Design TLB with 256 entry & two way set Associative for 40 bit VA, 36 bit physical Address, Page = 16KB