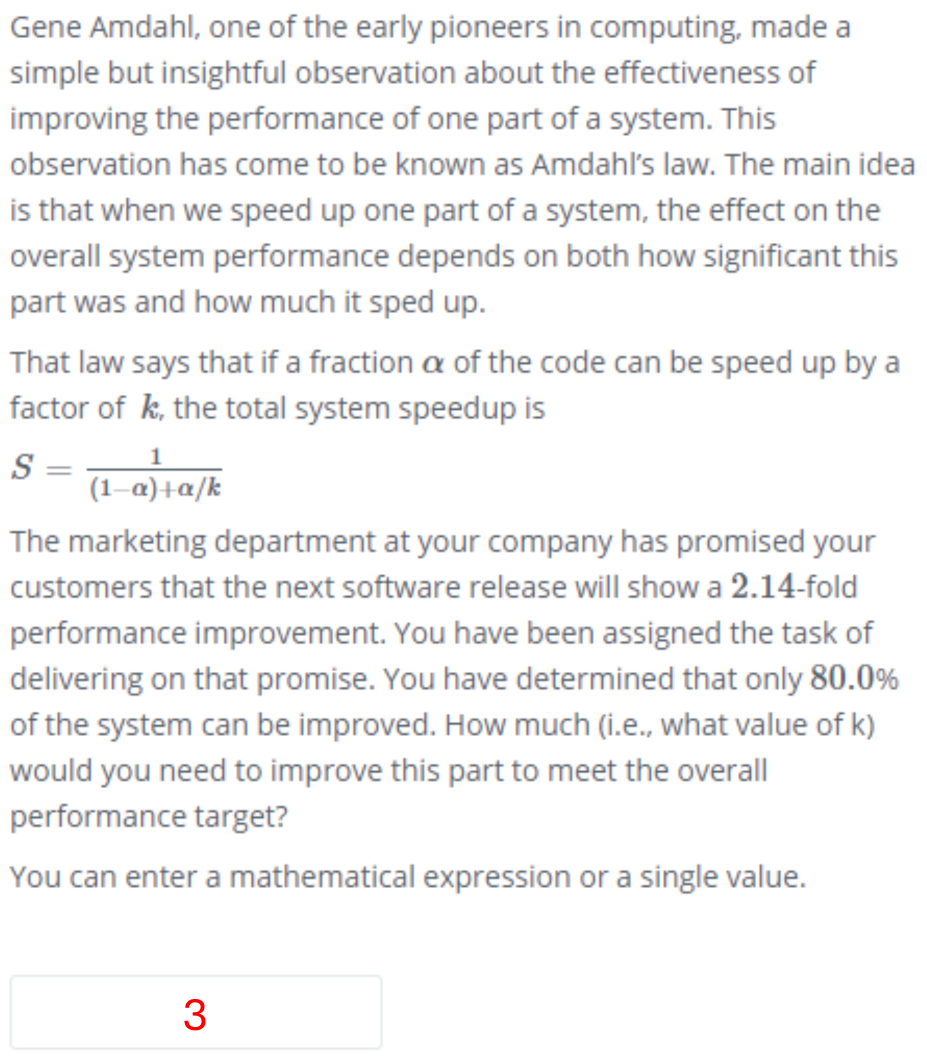
Question 1



Gene Amdahl, one of the early pioneers in computing, made a simple but insightful observation about the effectiveness of improving the performance of one part of a system. This observation has come to be known as Amdahl's law. The main idea is that when we speed up one part of a system, the effect on the overall system performance depends on both how significant this part was and how much it sped up.

That law says that if a fraction α of the code can be speed up by a factor of k, the total system speedup is

$$S = \frac{1}{(1-\alpha)+\alpha/k}$$

The marketing department at your company has promised your customers that the next software release will show a 2.14-fold performance improvement. You have been assigned the task of delivering on that promise. You have determined that only 80.0% of the system can be improved. How much (i.e., what value of k) would you need to improve this part to meet the overall performance target?

You can enter a mathematical expression or a single value.

3

- Amdahl's law

  $$S = \frac{1}{(1-\alpha)+\alpha/k}$$

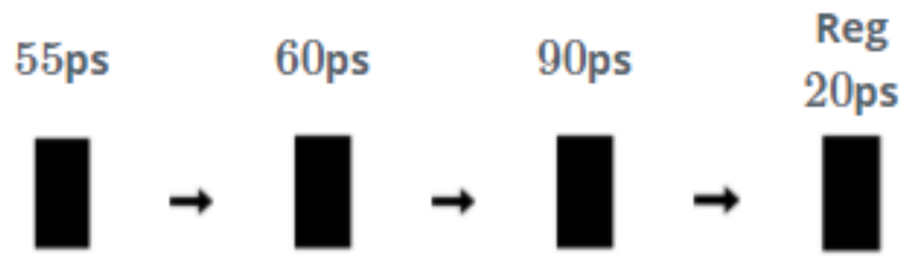
  - *S*: speedup of the execution of the whole task
  - *α*: proportion of execution time that the part benefitting from improved resources
  - *k*: speedup of the part of the task that benefits from improved system resources

In the given information, S=2.14, α=0.8

2.14 = 1 / ((1-0.8) + 0.8/k)
Therefore, k = 2.993... (≈ 3)

Question 2

We have a combinatorial logic function that can be decomposed into three steps each with the indicated delay with a resulting clock speed of 4.44 GHz.

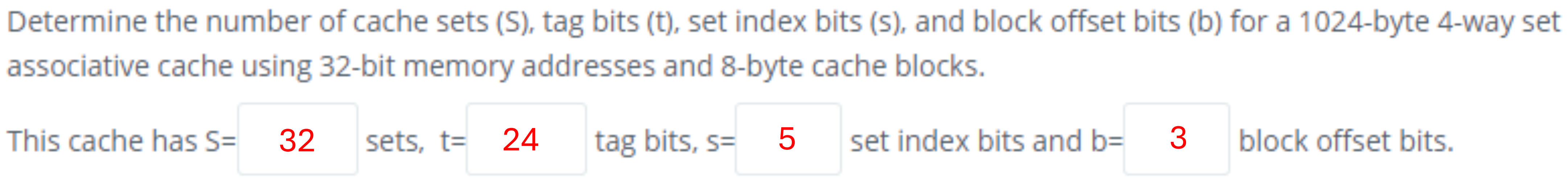


Assume we further pipeline this logic by adding just one additional register between the first two or last two stages of combinatorial logic. What would be the highest resulting clock speed we could achieve in GHz? 7.41

- Original delay = **225ps**
- Between first 2 stages
  - 55ps + 20ps = 75ps
  - 60ps + 90ps + 20ps = 170ps
- Between last 2 stages
  - 55ps + 60ps + 20ps = 135ps
  - 90ps + 20ps = 110ps

1/225ps : 1/135ps = 4.44GHz : x GHz
Therefore, x ≈ 7.41

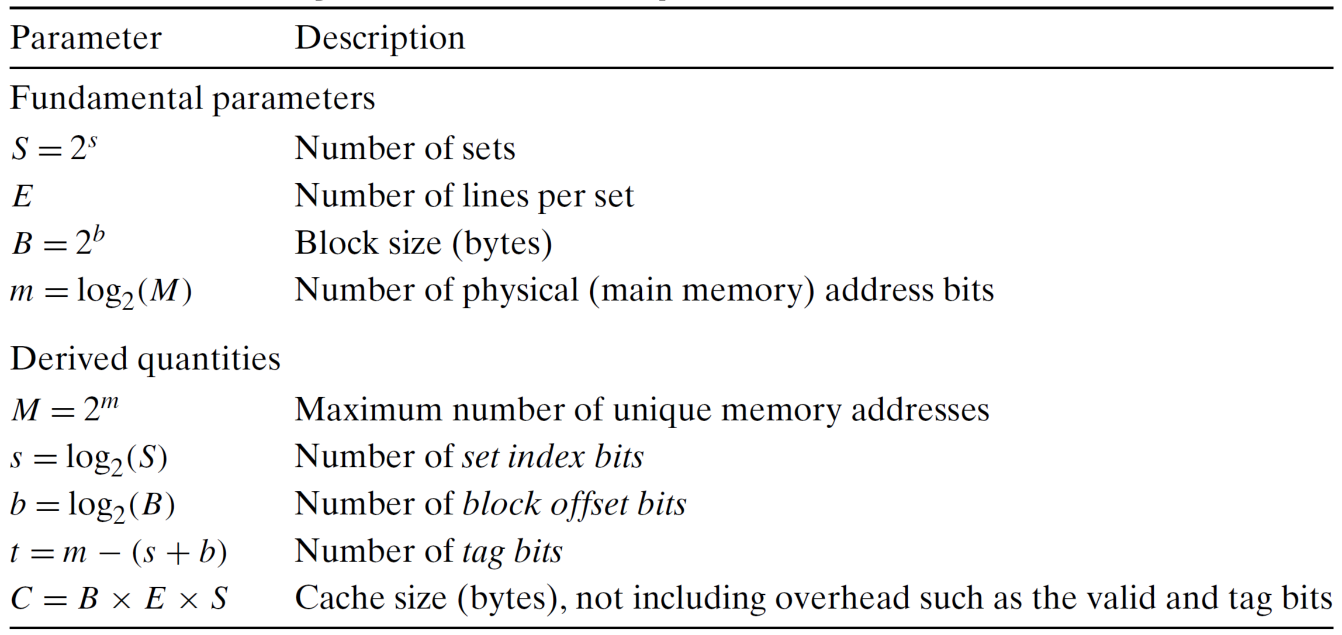
Question 3



Determine the number of cache sets (S), tag bits (t), set index bits (s), and block offset bits (b) for a 1024-byte 4-way set associative cache using 32-bit memory addresses and 8-byte cache blocks.

This cache has S= 32 sets, t= 24 tag bits, s= 5 set index bits and b= 3 block offset bits.

- Summary of cache parameters



Parameter	Description
Fundamental parameters	
$S = 2^s$	Number of sets
E	Number of lines per set
$B = 2^b$	Block size (bytes)
$m = \log_2(M)$	Number of physical (main memory) address bits
Derived quantities	
$M = 2^m$	Maximum number of unique memory addresses
$s = \log_2(S)$	Number of *set index bits*
$b = \log_2(B)$	Number of *block offset bits*
$t = m - (s + b)$	Number of *tag bits*
$C = B \times E \times S$	Cache size (bytes), not including overhead such as the valid and tag bits

- Number of sets (S) = 32

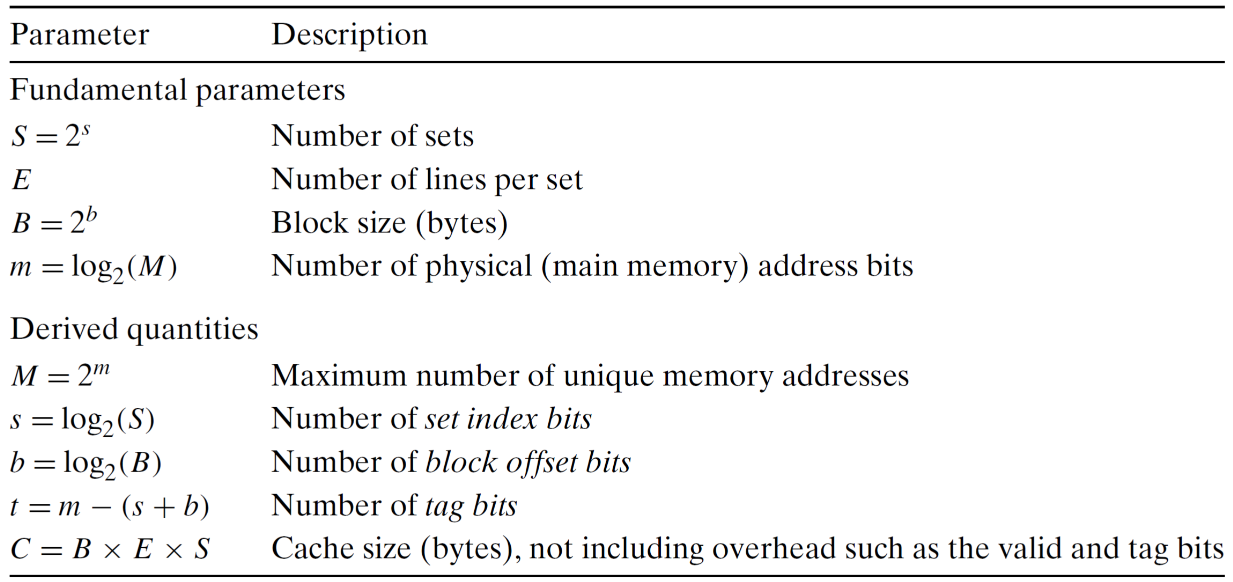
$$\frac{\text{total cache size (1024)}}{\text{\# of way set (4) * cache block size (8)}}$$

- Tag bits (t) = 24

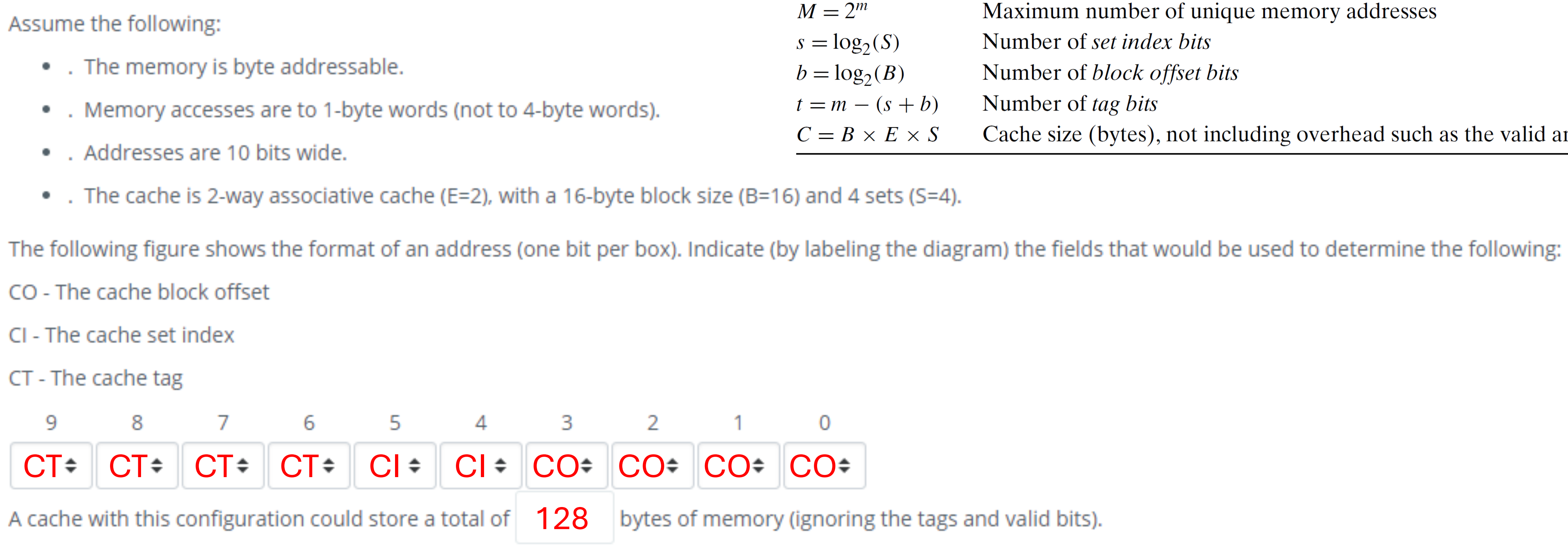
  32 bits memory addresses
  - (5 set index bits + 3 block offset bits)

- Set index bits (s) = 5
- Block offset bits (b) = 3

Question 4



Parameter	Description
Fundamental parameters	
$S = 2^s$	Number of sets
E	Number of lines per set
$B = 2^b$	Block size (bytes)
$m = \log_2(M)$	Number of physical (main memory) address bits
Derived quantities	
$M = 2^m$	Maximum number of unique memory addresses
$s = \log_2(S)$	Number of *set index bits*
$b = \log_2(B)$	Number of *block offset bits*
$t = m - (s + b)$	Number of *tag bits*
$C = B \times E \times S$	Cache size (bytes), not including overhead such as the valid and tag bits



Assume the following:

- . The memory is byte addressable.
- . Memory accesses are to 1-byte words (not to 4-byte words).
- . Addresses are 10 bits wide.
- . The cache is 2-way associative cache (E=2), with a 16-byte block size (B=16) and 4 sets (S=4).

The following figure shows the format of an address (one bit per box). Indicate (by labeling the diagram) the fields that would be used to determine the following:

CO - The cache block offset

CI - The cache set index

CT - The cache tag

9	8	7	6	5	4	3	2	1	0
CT	CT	CT	CT	CI	CI	CO	CO	CO	CO

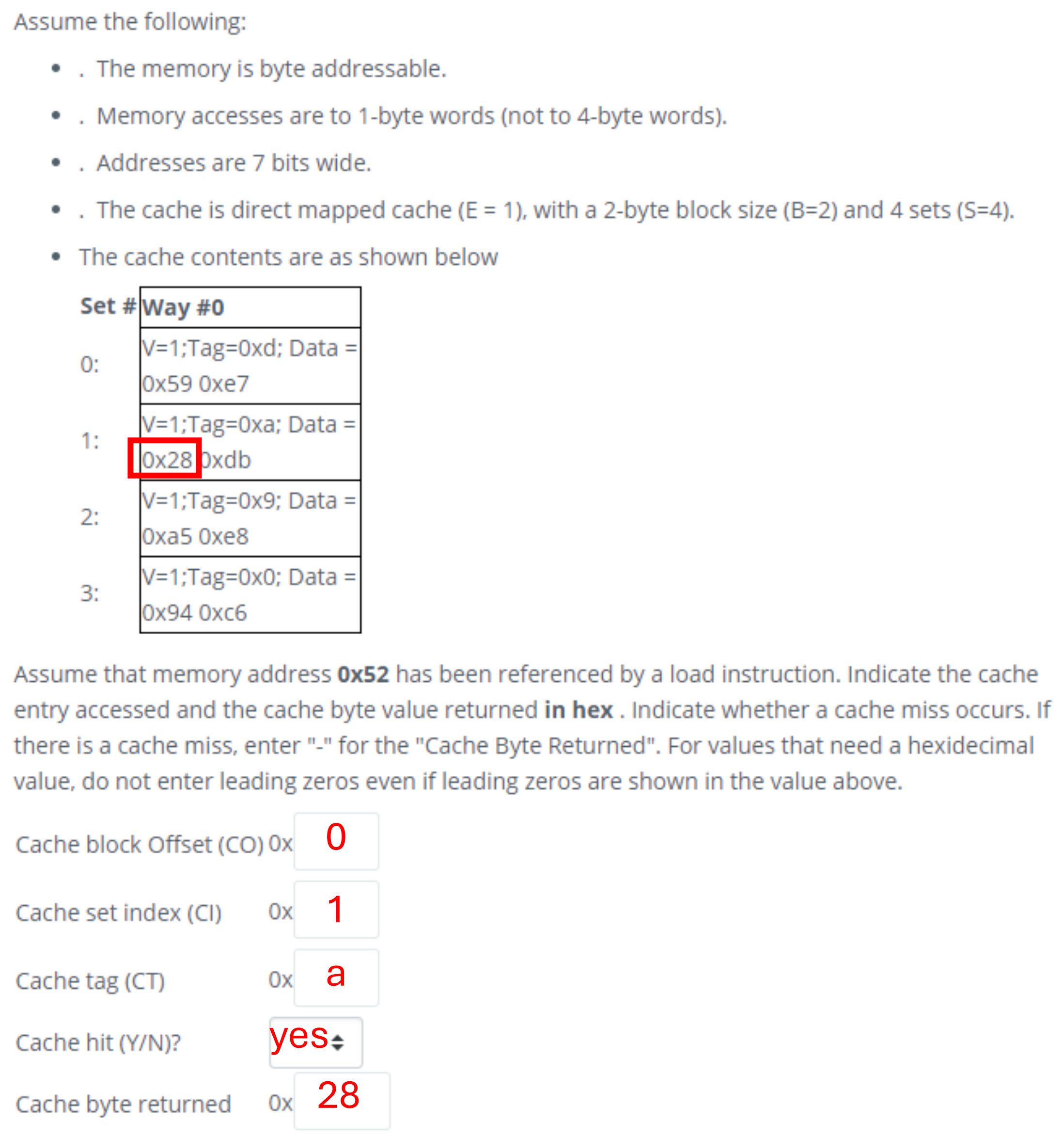
A cache with this configuration could store a total of 128 bytes of memory (ignoring the tags and valid bits).

- S=4, s=2 (2 CI)
- B=16, b=4 (4 CO)
- t=10-(2+4)=4 (4 CT)
- C=16x2x4=128 bytes in total

Question 5

- s=2(CI), b=1(CO), t=7-(2+1)=4(CT)
- 0x52 = 0101 0010$_{(2)}$
  - CO: 0x0
  - CI: 0x1
  - CT: 0xa

Parameter	Description
Fundamental parameters	
$S = 2^s$	Number of sets
E	Number of lines per set
$B = 2^b$	Block size (bytes)
$m = \log_2(M)$	Number of physical (main memory) address bits
Derived quantities	
$M = 2^m$	Maximum number of unique memory addresses
$s = \log_2(S)$	Number of *set index bits*
$b = \log_2(B)$	Number of *block offset bits*
$t = m - (s + b)$	Number of *tag bits*
$C = B \times E \times S$	Cache size (bytes), not including overhead such as the valid and tag bits



Assume the following:

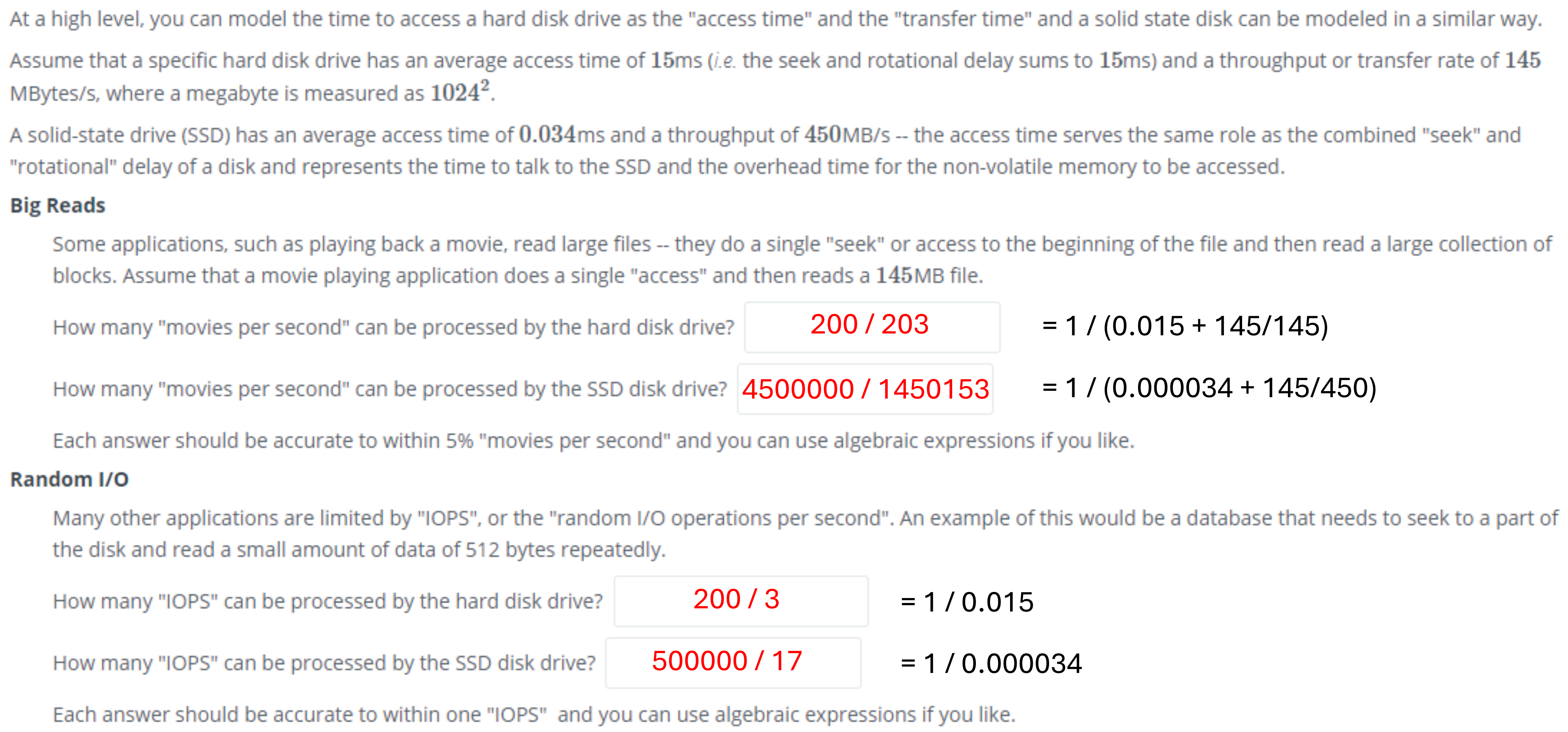
- . The memory is byte addressable.
- . Memory accesses are to 1-byte words (not to 4-byte words).
- . Addresses are 7 bits wide.
- . The cache is direct mapped cache (E = 1), with a 2-byte block size (B=2) and 4 sets (S=4).
- The cache contents are as shown below

Set #	Way #0
0:	V=1;Tag=0xd; Data = 0x59 0xe7
1:	V=1;Tag=0xa; Data = $\boxed{\text{0x28}}$ 0xdb
2:	V=1;Tag=0x9; Data = 0xa5 0xe8
3:	V=1;Tag=0x0; Data = 0x94 0xc6

Assume that memory address **0x52** has been referenced by a load instruction. Indicate the cache entry accessed and the cache byte value returned **in hex** . Indicate whether a cache miss occurs. If there is a cache miss, enter "-" for the "Cache Byte Returned". For values that need a hexidecimal value, do not enter leading zeros even if leading zeros are shown in the value above.

Cache block Offset (CO)	0x 0
Cache set index (CI)	0x 1
Cache tag (CT)	0x a
Cache hit (Y/N)?	yes
Cache byte returned	0x 28

Question 6



At a high level, you can model the time to access a hard disk drive as the "access time" and the "transfer time" and a solid state disk can be modeled in a similar way.

Assume that a specific hard disk drive has an average access time of 15ms (*i.e.* the seek and rotational delay sums to 15ms) and a throughput or transfer rate of 145 MBytes/s, where a megabyte is measured as 1024^2.

A solid-state drive (SSD) has an average access time of 0.034ms and a throughput of 450MB/s -- the access time serves the same role as the combined "seek" and "rotational" delay of a disk and represents the time to talk to the SSD and the overhead time for the non-volatile memory to be accessed.

Big Reads

Some applications, such as playing back a movie, read large files -- they do a single "seek" or access to the beginning of the file and then read a large collection of blocks. Assume that a movie playing application does a single "access" and then reads a 145MB file.

How many "movies per second" can be processed by the hard disk drive? 200 / 203 = 1 / (0.015 + 145/145)

How many "movies per second" can be processed by the SSD disk drive? 4500000 / 1450153 = 1 / (0.000034 + 145/450)

Each answer should be accurate to within 5% "movies per second" and you can use algebraic expressions if you like.

Random I/O

Many other applications are limited by "IOPS", or the "random I/O operations per second". An example of this would be a database that needs to seek to a part of the disk and read a small amount of data of 512 bytes repeatedly.

How many "IOPS" can be processed by the hard disk drive? 200 / 3 = 1 / 0.015

How many "IOPS" can be processed by the SSD disk drive? 500000 / 17 = 1 / 0.000034

Each answer should be accurate to within one "IOPS" and you can use algebraic expressions if you like.