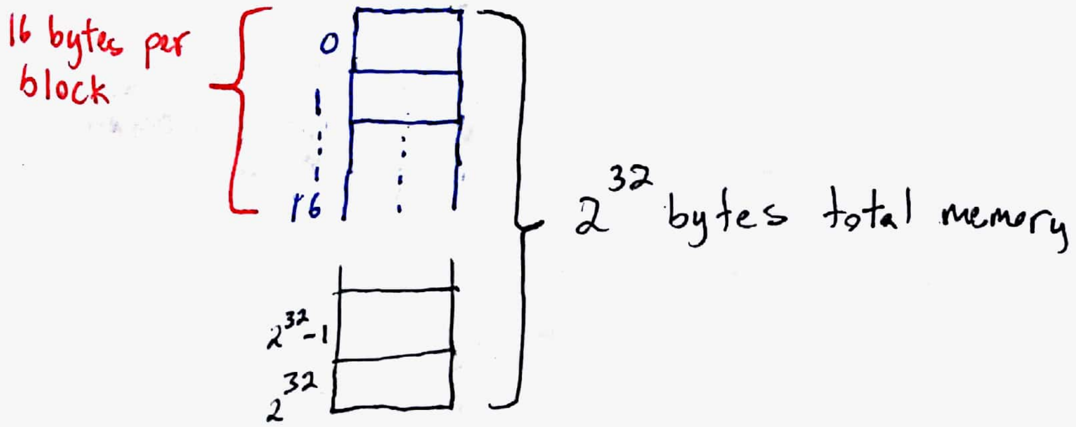
1 #1.)

a.) We have 2^{32} bytes of memory



$$\frac{2^{32} \text{ bytes}}{16 \text{ bytes}} = \# \text{ of cache Blocks}$$

↓

$$\frac{2^{32}}{2^{4}} = 2^{(32-4)} = 2^{28}$$

$$\boxed{\# \text{ of cache blocks} = 2^{28}}$$

1.)

b.)

Offset There are 2^4 bytes per block so we need 4 bits to represent all

Block There are 2^5 blocks so we need 5 bits to represent all

Answers

Tag The remaining bits go to tag. That is 32bits - 5bits - 4 bits which is 23 bits

#1.)

c.)

0x0000DB63

$$\underbrace{(0000)(0000)(0000)(0000)(1101)(101}_{\text{tag 23 bits}}\underbrace{1)(0110)}_{\text{block 5 bits}}\underbrace{(0011)}_{\text{offset 4 bits}}$$

10110

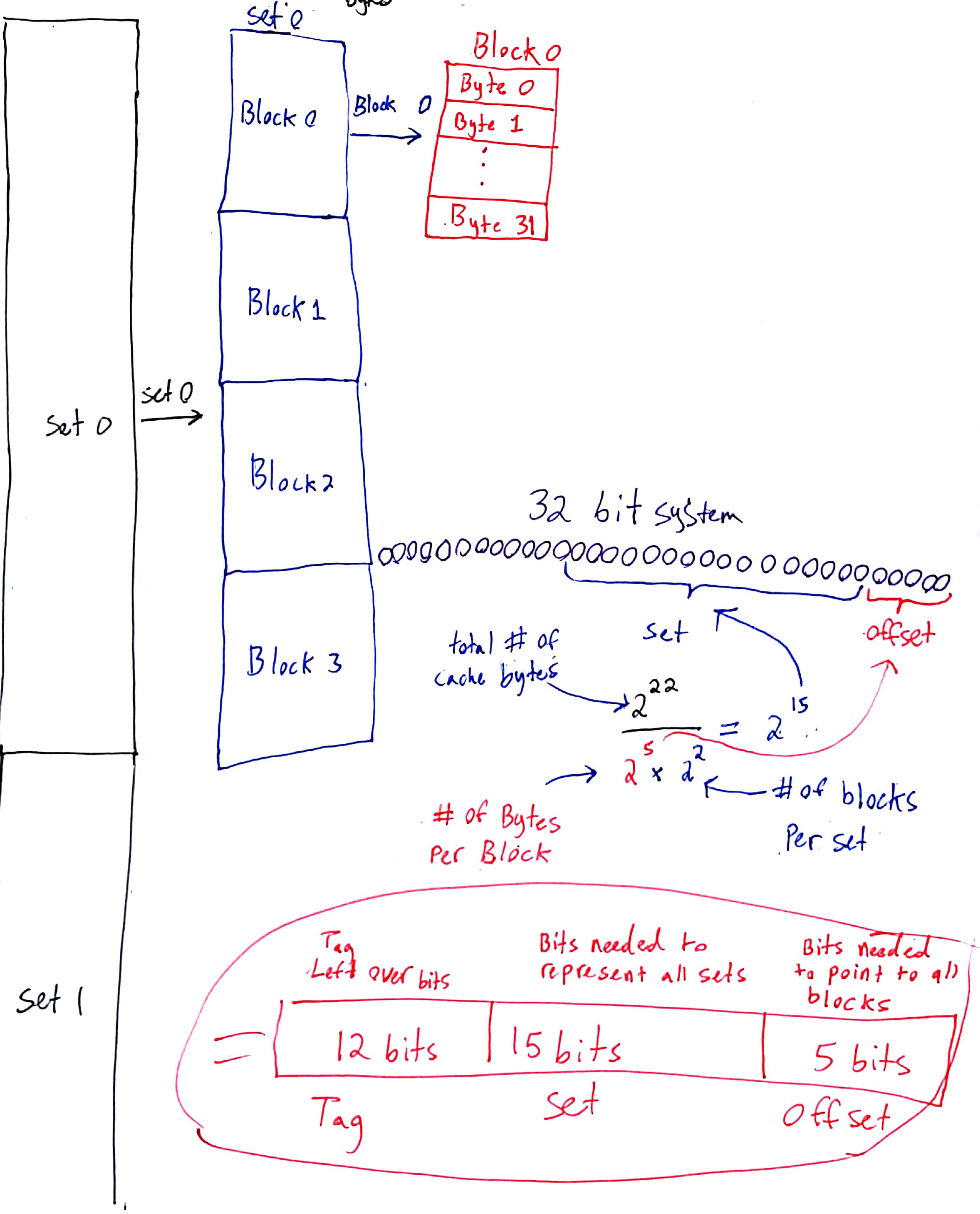
$2^4 + 2^2 + 2^1$

$16 + 4 + 2$

It maps to block 22

#2.)

4 MB = 2^{22} bytes



total # of cache bytes

$$\frac{2^{22}}{2^{5} \times 2^{2}} = 2^{15}$$

# of Bytes per Block

# of blocks Per set

Tag Left over bits	Bits needed to represent all sets	Bits needed to point to all blocks
12 bits	15 bits	5 bits
Tag	set	Offset

#3.) Since there are 16 bytes that need to be referenced the offset should have at least 4 bits.

Since there are 2^2 words per block and there are 2^5 blocks. That means we will need reference $2^5/2^2$ number of sets. There fore we will need 3 bits to represent all the sets.

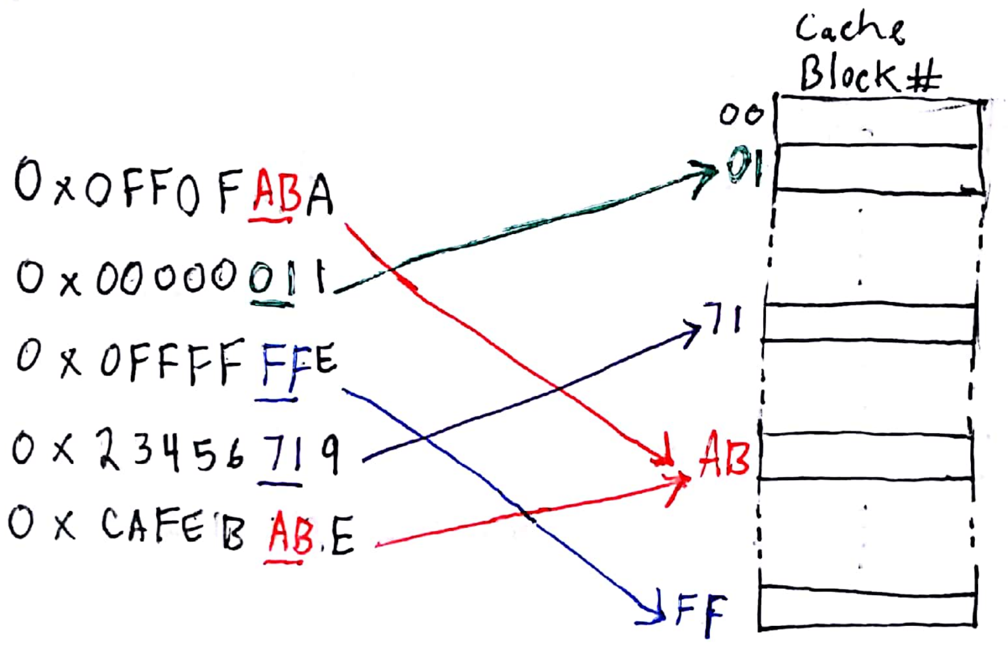
For total memory since we have 2^{18} total Bytes we have a total of 18 bits, so in order to figure out the tag bits, We will subtract the 3 bits for the set and the 7 bits for the offset. That means our tag bits will be 18-4-3 = 11 bits for tag.

Answer

Tag = 11 bits	Set = 3 bits	offset 4 bits

#4.)

Which if any of the addresses will cause a collision if they were accessed one after the other.



The first time we use a cache Block twice is at Cache Block 0xAB

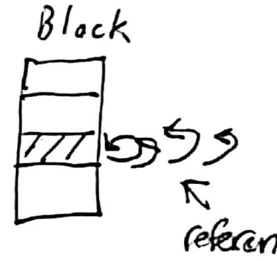
Answer collision at Cache Block 0xAB

#5.) Part 1

a.) Locality Principle

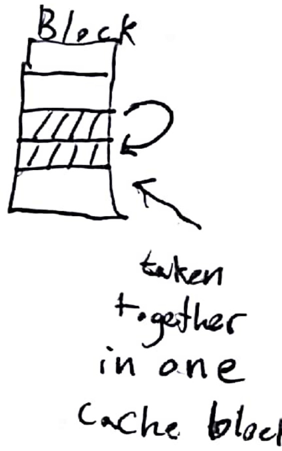
90% of execution time of the program is spent in 10% of the code.

b.) Temporal Locality



Recently referenced data is likely to be referenced again in the near future.

c.) Spatial Locality



Data with nearby addresses tend to be refrenced close together in time.

d.) Cache Valid bit

There is a bit at the end which indicates wether the data is good And can be executed.

e.) Cache Dirty bit

This is a way to identify if data brought from main memory has been modified by the cpu in cache. So the data matching it in the main memory can be overwritten to the new value.

#5.) Part 2

f.) Cache Tag

Cache tag is used to identify where a block of data must go from main memory. The size of the main memory modulo by the size of the cache block gives us the range of tag #s.

g.) Fully Associative Cache mapping

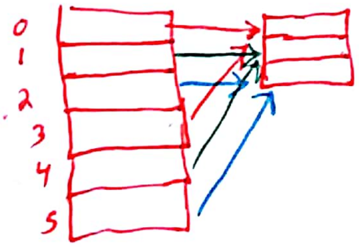
Any Block can be mapped to any cache Block. You can keep more frequently used cache block and evict less frequently used Cache blocks

32 bit
main memory
27 bit Tag

5 bit Word

h.) Direct Cache Mapping

Main memory blocks are mapped to corresponding factors of cache block



32 bit main memory
13 bit tag
14 bit slot
5 bit word

#5.) Part 3

i.) Set Associative Cache Mapping

This is like a mix of Direct Mapping and Associative Mapping. The sets are directly mapped. Inside each set there is associative mapping.

j.) LRU Replacement Algorithm

All slots are time stamped. When all slots are used up the first in the replacement queue is the slot that went the longest without being used.

k.) LFU Replacement Algorithm

Every time a slot is used we increment the counter. Then when all slots are used up. We evict the slot that has the lowest counter Value

#5.) Part 4.)

L.) Random Replacement Algorithm

When full a random slot gets replaced.

M.) Write Through Policy

When the cpu writes to the cache it also writes to the main memory. This makes eviction faster.

N.) Write Back Policy

Writes to cache only until the slot is getting evicted. Then it writes to the main memory also.

O.) Write Allocate

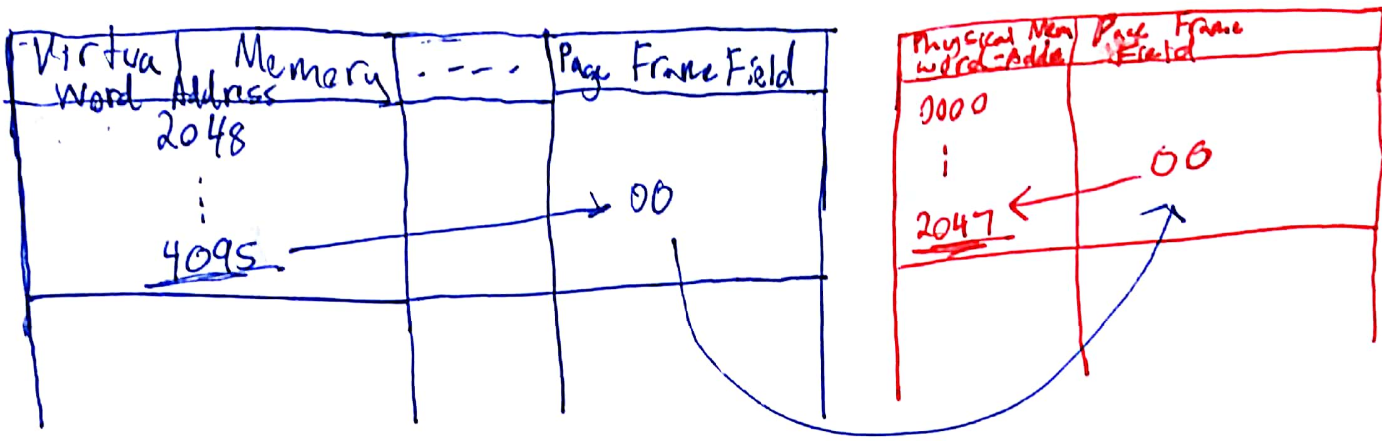
writes data to cache then uses write through or writeback policies. This is good for subsequent writes to the same slot.

P.) Write No-Allocate

When writing, if data is not found in Cache. Then we update the data in main memory but not in cache.

F6.)

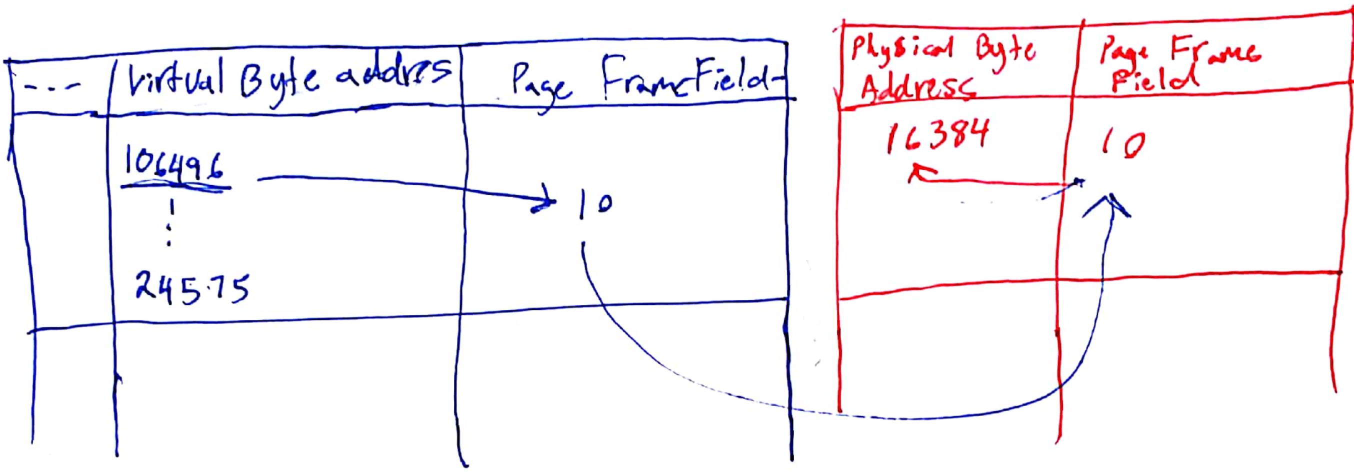
a.)



Virtual Memory Word Address	- - -	Page Frame Field
2048 : 4095		00

Physical Mem word Addr	Page Frame Field
0000 : 2047	00

b.)



- - -	Virtual Byte addres	Page Frame Field
	106496 : 24575	10

Physical Byte Address	Page Frame Field
16384	10