Exercise 4: Given a computer with 32-bit addresses, with a no-write allocate direct-mapping 128B cache, with blocks of 16B. Assume that variable *a* is stored in the register file.

```
int nota[128];        // nota[0] is stored in address 0x00000000
int media[128];       // media[0] is stored after nota[127]

for (i=0;i<128;i++) {
            if (i>7 && i<64) {
                        nota[i] = media[i]/2;
                        }
            else {
                        a = nota[i]*media[i]
                        }
}
```

a) Obtain the amount of misses

b) Propose 2 code optimizations and obtain the amount of misses.

c) Assume that: time to access a word in cache is 1ns, time to read or write a word in main memory is 50ns and time to transfer a MM block to cache is 200ns. What is the time needed for all the references in the program?

d) If a second level cache memory with the following features is added to the computer: 1MB, 4-ways, block of 16B, with no-write allocate and write-back policies. Assuming the time to transfer a block from the second level cache to the first level one is 15ns, what would be the time to access all the references in the program?

1) Memory accesses

```
int nota[128];        // nota[0] is stored in address 0x00000000
int media[128];       // media[0] is stored after nota[127]

for (i=0;i<128;i++) {
        if (i>7 && i<64) {
                nota[i] = media[i]/2;
                }
        else {
                a = nota[i]*media[i]
                }
}
```

From i=8 to i=63 (56 iterations): 1 memory read (media) and one memory write (nota)

The remaining 64+8 iterations: Two memory reads (nota and media)

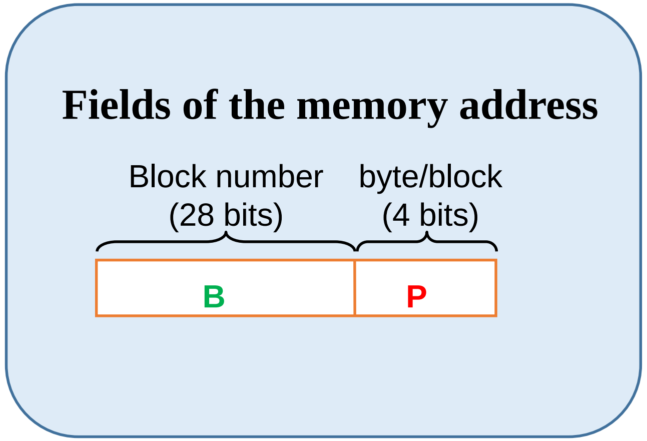
Accesses for fetching instructions are not considered in this exercise.

2) Memory layout and blocks

Blocks of 16B

$$\frac{16\ bytes/block}{4\ bytes/element} = 4\ elements/block$$

Each array needs 32 memory blocks



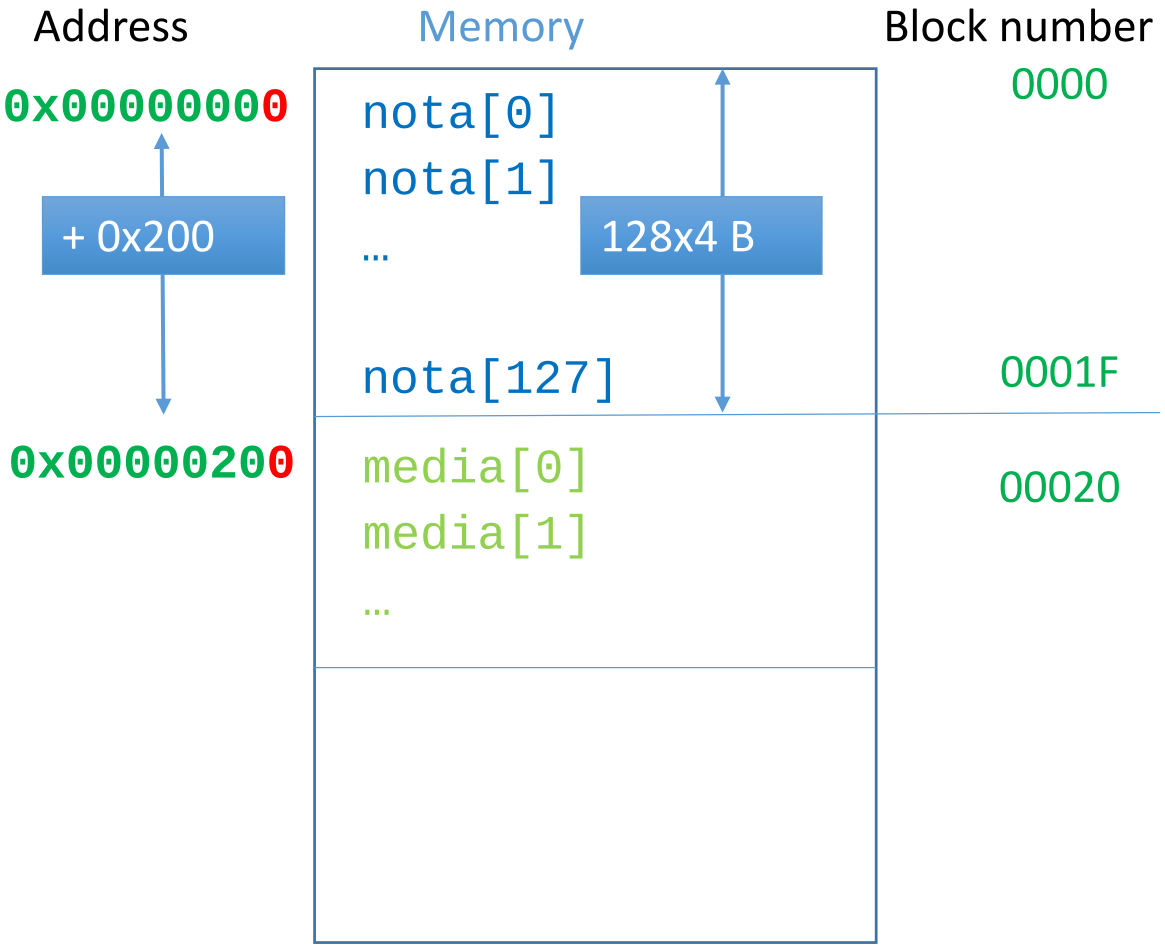
Fields of the memory address

Block number (28 bits)

byte/block (4 bits)

B

P



Address

Memory

Block number

0x00000000

+ 0x200

0x00000200

nota[0]

nota[1]

…

nota[127]

128x4 B

media[0]

media[1]

…

0000

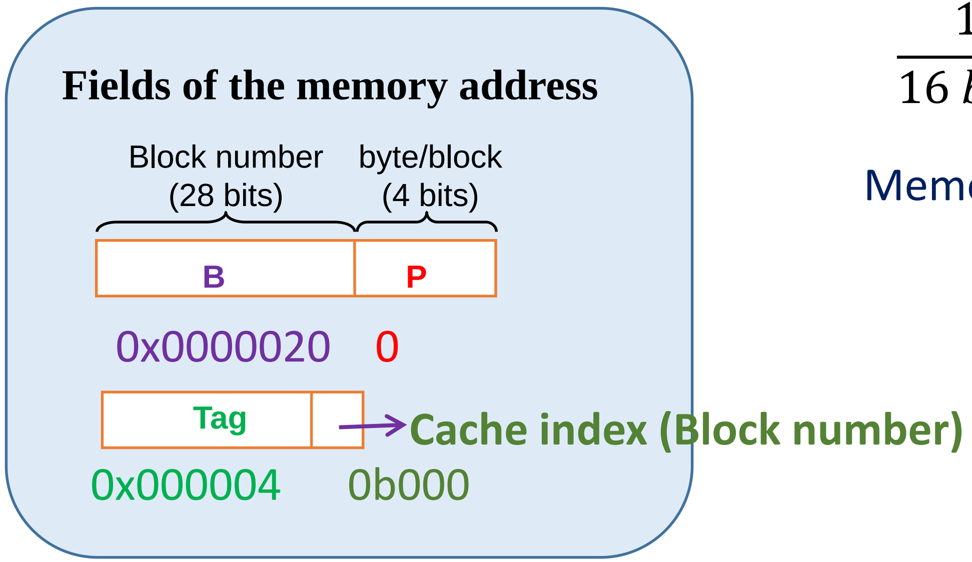
0001F

00020

3) Cache structure

Direct mapped cache memory of 128B with block of 16B

Where in the cache is media[0] stored?



Fields of the memory address

$$\frac{128\ bytes}{16\ bytes/block} = 8\ blocks$$

Memory Address: 0x00000200

0x0000020

$$\overbrace{\underbrace{0000\ 0000\ 0000\ 0000\ 0000\ 0010\ 0}_{0x000004}\quad 000}^{0x0000020}$$

For the same value of i, nota and media map to the same cache block.

Accesses and misses (1)

```
for i=0 to i=7 and for i=64 to 127
        a = nota[i]*media[i]
```

Iterations: 64 + 8 = 72

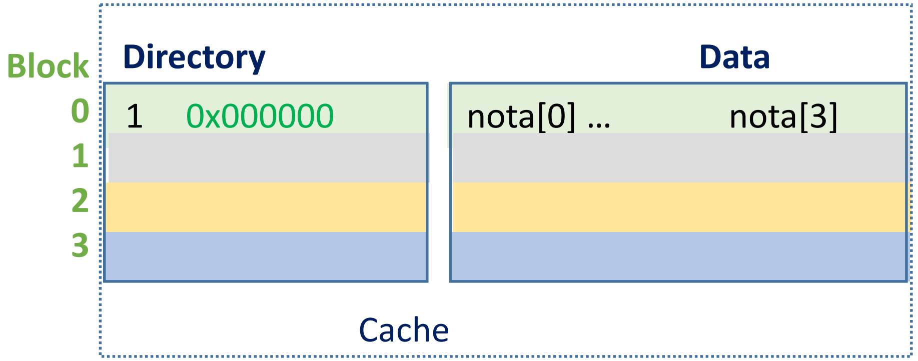
Accesses: 2 per iteration
(reads)

Misses: 2 per iteration
(read_misses)

i=0

1. Read nota[0]

Cache miss

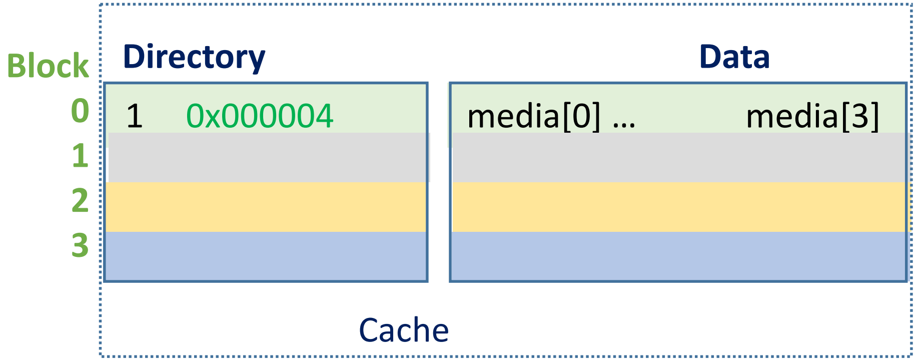


Block	Directory		Data	
0	1	0x000000	nota[0] …	nota[3]
1				
2				
3				

Cache

2. Read media[0]

Cache miss



Block	Directory		Data	
0	1	0x000004	media[0] …	media[3]
1				
2				
3				

Cache

Conflict:
2 memory blocks are mapped to the same cache block

Accesses and misses (2)

```
for i=8 to 63
        nota[i] = 0,5*media[i]
```

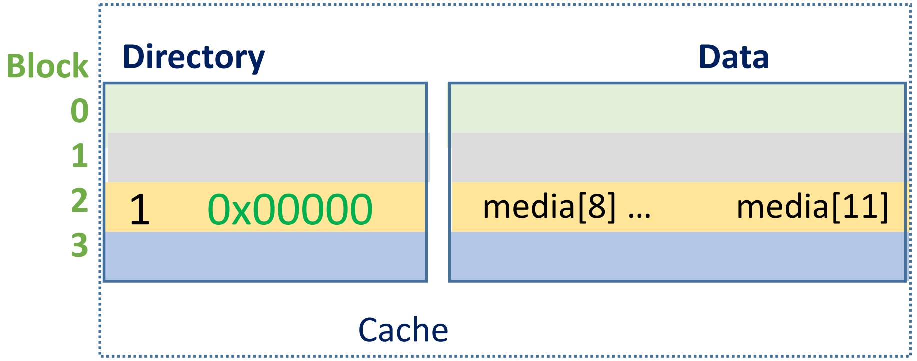
i=8

1. Read media[8]

   Cache miss

2. Write nota[8]

   Cache miss: no-write allocate



Block	Directory	Data
0		
1		
2	1 0x00000	media[8] … media[11]
3		

Cache

Iterations: 64 - 8 = 56

Accesses: 1 read + 1 write per iteration

Misses: 1 read miss every four iterations + 1 write miss per iteration

One read miss per block.
All writes are misses (elements are writen in memory)

a) Obtain the amount of misses

Iterations: 64 + 8 = 72

Accesses: 2 per iteration

Misses: 2 per iteration

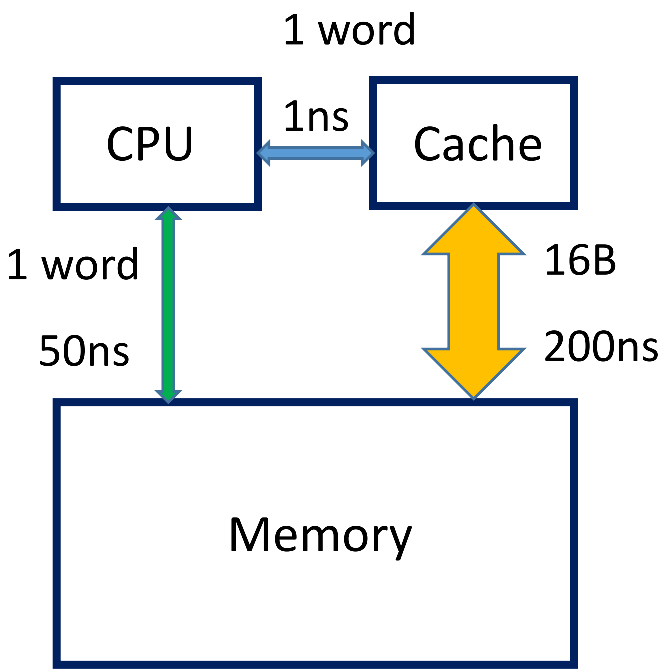
Iterations: 64 - 8 = 56

Accesses: 1 read + 1 write per iteration

Misses: 1 read miss every four iterations + 1 write miss per iteration

# Accesses = 2 * 128 = 256
# Read_misses = 2 * 72 + 56/4 = 158
# Write_misses = 1 * 56

c) Assume that: time to access a word in cache is 1ns, time to read or write a word in main memory is 50ns and time to transfer a MM block to cache is 200ns. What is the **time needed for all the references in the program**?



Time_mem = #Accesses * T_hit + #Read_misses * Penalty_read + #Write_misses * Penalty_write

#Accesses = 256
#Read_misses = 158
#Write_misses = 56

#Penalty_read: read block from MM
#Penalty_write: write word to MM

Time_mem = 256 * 1ns + 158 * 200ns + 56 * 50ns = 34656ns

If we need average access time (**AMAT**):

1st option:

Average_Time_mem = Time_mem / #Accesses = 135,4ns

2nd option:

Average_Time_mem = T_hit +
(#Read_misses/ #Accesses) * Penalty_read +
(#Write_misses/ #Accesses) * Penalty_write

#Read_miss_ratio = (#Read_misses/ #Accesses) = 0,6172
#Write_miss_ratio = (#Write_misses/ #Accesses) = 0,21875

Average_Time_mem = 1ns + 0,6172 * 200ns + 0,21875 * 50ns

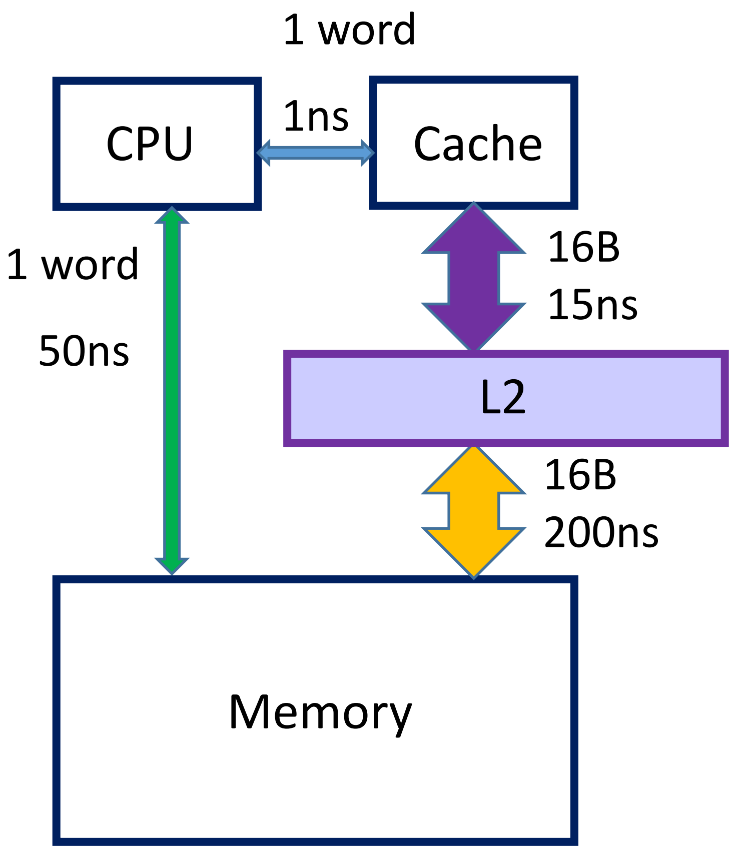
d) If a second level cache memory with the following features is added to the computer: 1MB, 4-ways, block of 16B, with no-write allocate and write-back policies. Assuming the time to transfer a block from the second level cache to the first level one is 15ns, what would be the time to access all the references in the program?

Time_mem = #Accesses * T_hit + #Read_misses * Penalty_read + #Write_misses * Penalty_write

Can be modified by L2!

#Penalty_read: read block from L2

#Penalty_write: write word to MM (we will consider it doesn't change)



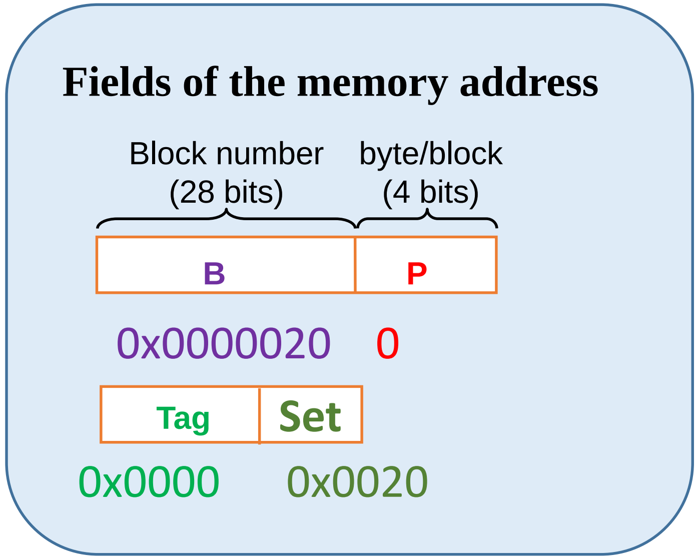
L2 Cache structure

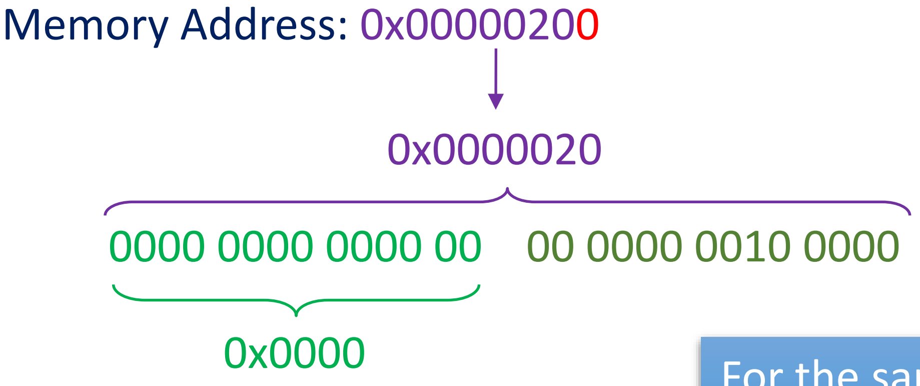
4-way set associative cache memory of 1MB with block of 16B

Where in the cache is media[0] stored?

$$\frac{2^{20}\ bytes}{16\ bytes/block} = 2^{16}\ blocks$$

$$\frac{2^{16}\ blocks}{4\ ways/set} = 2^{14}\ sets$$





For the same value of i, nota and media map to different sets.
No conflicts!

Accesses and misses to L2 (1)

Only L1 misses access L2

for i=0 to i=7 and for i=64 to 127
a = nota[i]*media[i]

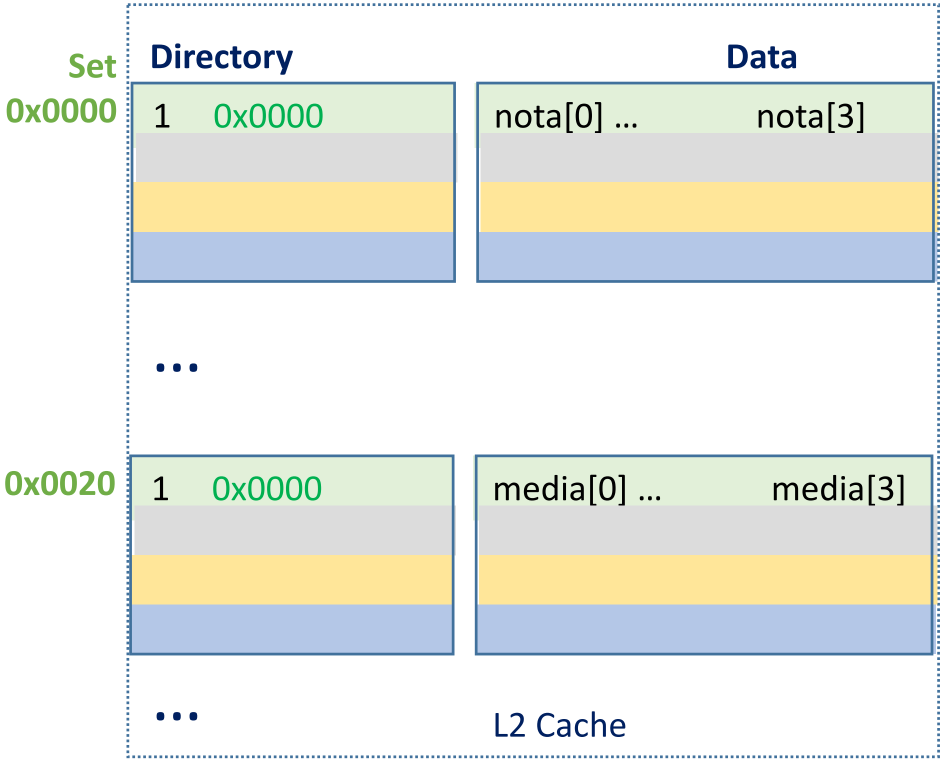
i=0

1. Read nota[0]

   L1 cache miss
   L2 cache miss (first access)

2. Read media[0]

   L1 cache miss
   L2 cache miss (first access)



Set

Directory **Data**

0x0000 1 0x0000 nota[0] … nota[3]

…

0x0020 1 0x0000 media[0] … media[3]

…

L2 Cache

Accesses and misses to L2 (2)

Only L1 misses access L2

```
for i=0 to i=7 and for i=64 to 127
          a = nota[i]*media[i]
```

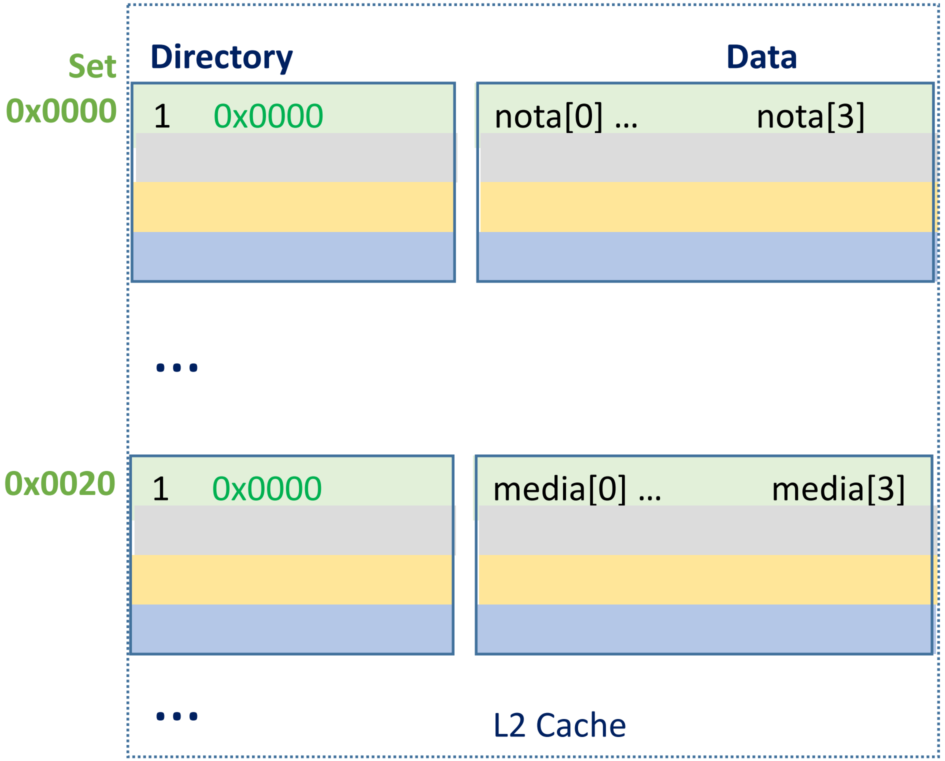
i=1

1. Read nota[1]

   L1 cache miss
   L2 cache hit

2. Read media[1]

   L1 cache miss
   L2 cache hit



Set

Directory

Data

0x0000

1 0x0000

nota[0] … nota[3]

…

0x0020

1 0x0000

media[0] … media[3]

…

L2 Cache

Iterations: 72

Accesses: 2 per iteration

Misses: 2 every 4 iterations

Accesses and misses to L2 (3)

Only L1 misses access L2

```
for i=8 to 63
        nota[i] = 0,5*media[i]
```

i=8

1. Read media[8]

   L1 cache miss
   L2 cache miss (first access)

2. Write nota[8]

   L1 cache miss: no-write allocate
   L2 cache miss

One read miss per block.
All writes are misses (elements are writen in memory)

Iterations: 64 - 8 = 56

Accesses: 1 read every 4 iterations (i = 8, 12, 16…) + 1 write per iteration

Misses: 1 read miss every four iterations + 1 write miss per iteration

#Read_misses * Penalty_read (time for all reads to L2):
L2_Reads_time =# L2_reads * L2_Hit_time + #L2_Read_misses * L2_Penalty_read

Iterations: 64 + 8 = 72

Accesses: 2 per iteration

Misses: 2 every 4 iterations

Iterations: 64 - 8 = 56

Accesses: 1 read miss every 4 iterations

Misses: 1 read miss every 4 iterations

# L2_reads = # L1_Read_misses = 2 * 72 + 56/4 = 158
# L2_Read_misses = (2 * 72)/4 + 56/4 = 50

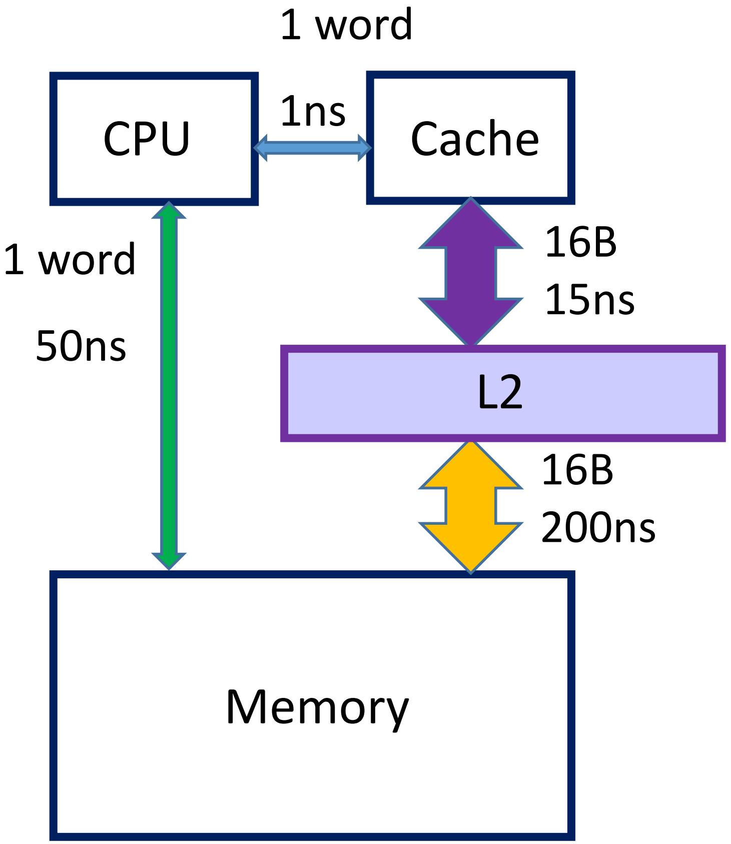
#Read_misses * Penalty_read (time for all reads to L2):
L2_Reads_time =# L2_reads * L2_Hit_time +
#L2_Read_misses * L2_Penalty_read

# L2_reads = 158
# L2_Read_misses = 50

L2_Hit_time = 15ns
#L2_Penalty_read = 200ns

L2_Reads_time = 158 * 15ns + 50 * 200ns = 12370ns

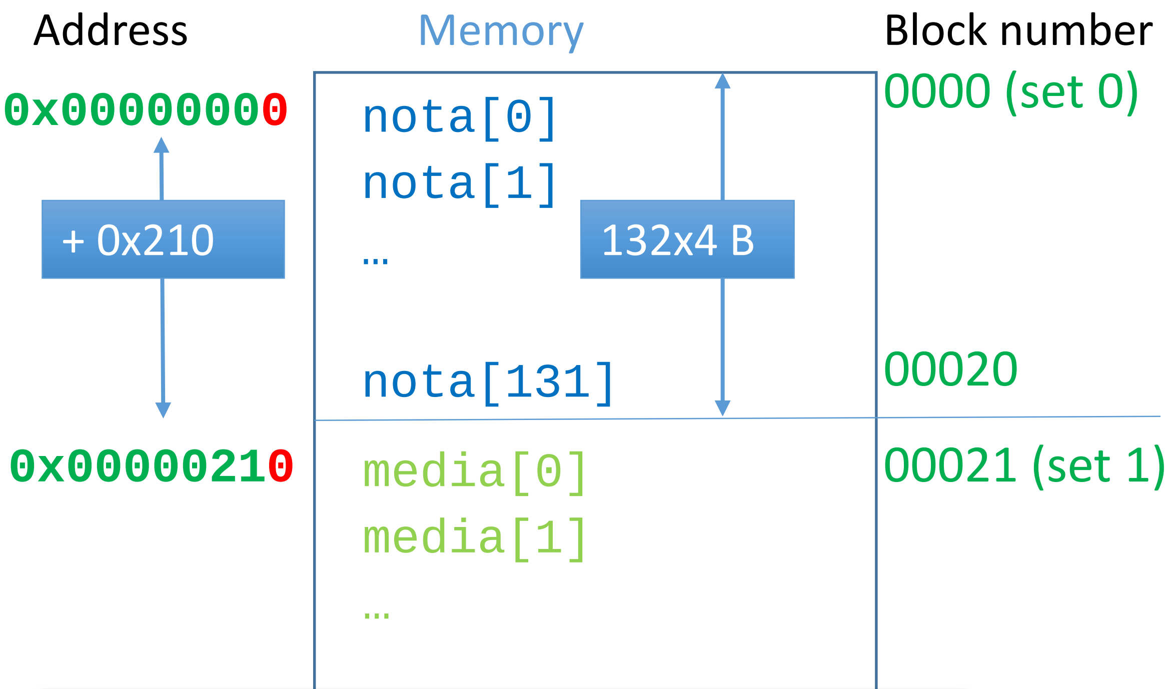
Time_mem = 256 * 1ns + 12370ns + 56 * 50ns = 15426ns



Code optimizations

1. **Array enlargement**: **add an empty block** to the first array (nota) so that the second array maps to a different set.

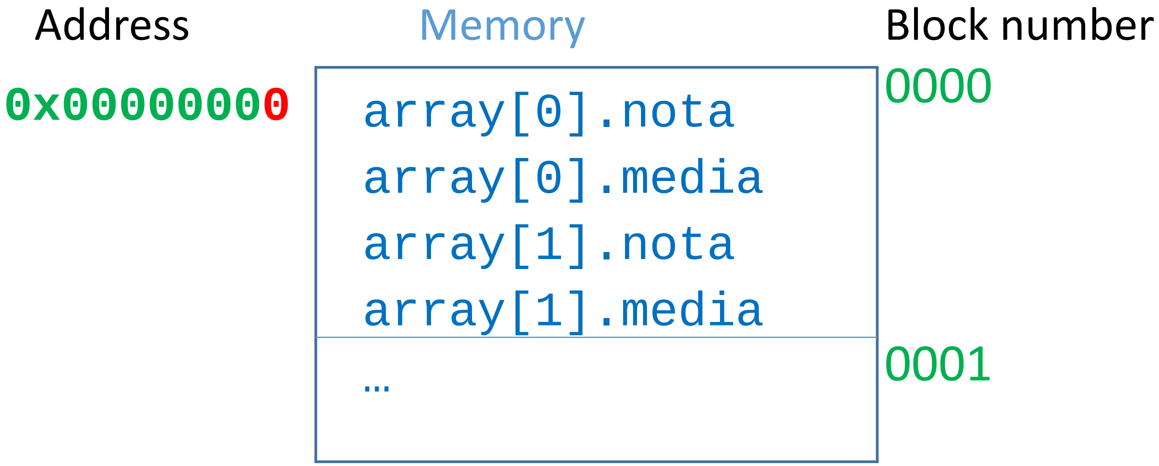
```
int nota[132];
```



Read misses are reduced to compulsory (50)

2. **Merge arrays: place the same positions of different arrays** in contiguous memory locations.

```
struct fusion{
        int nota;
        int media;
} array[128];
```



Read misses are reduced to compulsory (64), write misses disappear