Memory and cache hierarchy
Solved exercises

Exercise 1. Be a 32-bit computer with a cache of 256 KB, lines of 64 bytes and an access time of 5 ns. The cache is 4-way associative, and it is used the LRU replacement policy.
It is requested:

a) Indicate the number of lines and sets in the described cache.
b) What is the size of the blocks that are transferred between the cache and the main memory?
c) If the time to transfer a block from main memory to cache is 200 ns, indicate the required cache hit rate, so that the average time to access the memory system is 20 ns.

Solution:

a) The cache has a size of 256 KB = 2^{18} bytes. As each line has 2^6 bytes, the number of lines is 2^{18} bytes / 2^6 bytes = 2^{12} lines = 4096 lines. As the cache is 4-ways associative, each set has four lines, therefore the number of sets is 4096 / 4 = 1024 sets.
b) The size of the block that is transferred between main memory and cache matches the size of the line, that is, 64 bytes.
c) The average time of access to the system is given by the following expression:

$tm = tc \cdot Pa + (1-Pa) \cdot tf$

where tc = 5 ns, tm = 20 ns and tf = 205 (200 + 5). Therefore:

$20 = 5 \cdot Pa + (1-Pa) \cdot 205$

Clearing, it is obtained that Pa = 185 / 200 = 0.92. That is, the hit rate must be 92%.

Exercise 2. A computer is available with memory addresses of 32-bits, addressing memory by bytes. The computer has an associative cache memory of 4-ways, with a line size of 4 words. This cache has a size of 64KB. The cache access time is 2 ns, and the time needed to deal with a cache miss is 80 ns.

Indicate in a reasoned manner:

a) Size in MB of the memory that can be addressed on this computer.
b) The number of words that can be cached.
c) The number of lines that can be stored in the same set.
d) Number of lines in the cache.
e) Number of sets in the cache.
f) Indicate the hit rate necessary for the average time of access to the memory system of this computer to be 10 ns.

Solution:

a) If the computer has 32-bit memory addresses the main memory size will be 2^{32} bytes = 2^{12} MiB
b) Cache size = 64 KiB
As the computer is 32 bits, the words are 32 bits, 4 bytes, so the number of words that the cache can store is 64 KB / 4 bytes = 16 * 1024 words.
c) 4 lines can be stored in each set.
d) The line size is 4 words -> 4*4 = 16 bytes. Number of lines = 64 KB / 16 bytes = 2^{16} / 2^4 = 2^{12} = 4096 lines
e) The number of cache sets will be the number of cache lines divided by the number of lines in each set = 4096 lines / 4 lines = 1024 sets
f) Tm = Pa * T_cache + (1-Pa) * T_fallo = > 10 = Pa*2 + (1-Pa) * 80 = > Ta = 70 / 78, i.e. a hit rate of 89%

Exercise 3. According to a study carried out on the use of the instructions of this computer it has been determined that on average it executes 50 million instructions per second and that the percentage of use of its instructions is as follows:

LOAD	30 %
STORE	10 %
MOVE	10 %
Arithmetic operations	24 %
Logical operations	6 %
Branches	20 %

It is requested:

a) Determine the number of memory accesses per second that are performed on this computer.
b) What will be the number of main memory accesses per second if you use a cache with a line size of 8 words, write-through update policy and a hit rate of 95%?

Solution:

a) The computer executes 50 million instructions per second. The execution of each instruction implies an access to reading memory for the reading of the instruction itself. In addition, of all these, 40% (corresponding to the LOAD and STORE instructions) involve additional access to main memory. Therefore, in one second, 50 million memory accesses are made per second plus 20 million (40% of 50) accesses for the LOAD and STORE instructions.

   The total number of memory accesses is 70 million accesses per second.

b) Of all the 70 million memory accesses, it is necessary to determine how many are read and how many are write. 50 million are readings of the instructions themselves. Of the 20 million memory accesses corresponding to the LOAD and STORE instructions, 75% (3 out of 4) correspond to reads of the LOAD statement and the remaining 25% to writes corresponding to STORE instructions, that is, 15 million accesses are read and 5 million access per second are write.

   The total number of reads per second = 50 +15 = 65 million read accesses per second.
   The total number of writes per second = 5 million write accesses per second.

   Of the total read accesses, 5% represent misses that must be resolved through their corresponding access to main memory. Therefore, the number of misses per second to main memory for readings will be 0.05 * 65 = 3.25 million. Remember that each cache miss involves the reading of a complete block, which in this case is 8 words, therefore, the number of accesses that occur to main memory because of read misses = 8 * 3.25 = 26 million accesses per second

   Of the total write accesses, all involve main memory accesses because an immediate write policy is used. Assuming that the block causing the miss is modified at the lower level (main memory) and is not loaded into the cache, all writes access main memory. Therefore, the number of write accesses will be 5 million accesses per second.

   The total number of main memory accesses per second = 5 + 26 = 31 million accesses per second.

Exercise 4. Be a 32-bit computer with a data cache of 32 KB and 64-byte lines. The cache is associative with 2-way sets and the LRU replacement policy is used. Given the following code snippet:

```
int v[262144]

for (i = 0; i < 262144; i = i + 2)
      v[i] = 9;
```

It is requested:

a) Describe the number of lines and sets in the data cache described in the statement.
b) Considering only the data cache and the accesses to the vector, calculate in a reasoned way the miss rate that is obtained in the execution of the previous loop.

Solution:

c) The cache is 32 KB = 2^{15} bytes in size. as each line has 2^6 bytes, the number of lines is 2^{15} bytes / 2^6 bytes = 2^9 lines = 512 lines. As the cache is associative by 2-way sets, each set has two lines, therefore the number of sets is 512 / 2 = 256 sets

d) The pattern of vector accesses to the loop is as follows:

v[0], v[2], v[4], v[6], v[8] ...

that is, every 2 elements are accessed. Because the cache is structured in 64-byte lines and each integer (int) data takes up 4 bytes, each line accommodates 16 elements of the vector. As the vector is traversed sequentially and of these 16 only 8 are actually accessed, the miss rate is 1/8.

Exercise 5. Be a 32-bit computer that has a cache of 512 KB, associative by sets of 4 ways and lines of 64 bits. On this computer you want to run the following code snippet:

```
int v[200];
int i;
int s;

for (i=0; i < 200; i++)
      s = s + v[i];
```

It is requested:

a) Reasonably indicate the number of lines and sets in the cache.
b) If the cache is considered empty and the values of the variables i and s of the previous code are stored in registers, indicate, considering only the accesses to the vector v, the hit rate that is obtained when executing the previous code fragment.

Solution:

a) The cache has a size of 512 KiB = 2^{19} bytes. Each line has 64 bits = 8 bytes = 2^3 bytes. The line number is given by 2^{19} bytes / 2^3 bytes = 2^{16} lines. Since each set has 4 lines, the number of sets = 2^{16} / 2^2 = 2^{14} sets

a) Since each element of the vector is a 4-byte integer, each line fits two integers. Each time an item is accessed, a miss occurs, and the item that caused the miss and the next one, which results in a success on the next loop turn, are cached. Therefore, there is a miss every two accesses and the hit rate is 50%.

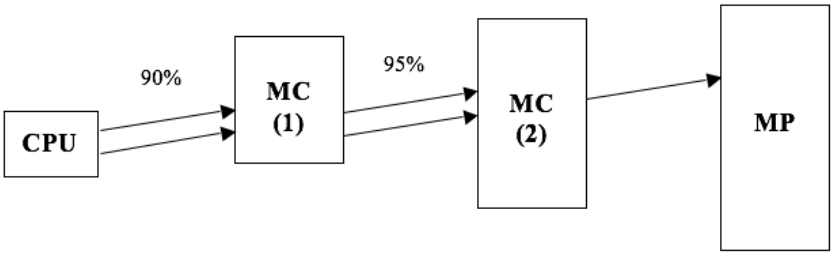
Exercise 6. A system with a 2-level cache is available. When running a certain application, the hit rate of the level 1 cache is 90% and the hit rate of the level 2 cache is 95%. The application generates one million memory accesses during its execution. Indicate in a reasoned manner:

a) The number of accesses that are generated to the level 1 cache.
b) The number of accesses that are generated to the level 2 cache.
c) The number of accesses that are generated to main memory.

Solution:

a) The number of successful accesses in CM(1) is: **10^6**
b) The number of successful accesses in CM(2) is: **10^6 * 0.1**
c) The number of accesses with access to MP due to MC1 and MC2 failures: **10^6 * 0.1 * 0.05**

The following figure illustrates how it works.



Exercise 7. Be a 32-bit computer that has a cache of 512 KB, associative by sets of 4 ways and lines of 128 bytes. On this computer you want to run the following code snippet:

```
int a1[200];
int a2[200]:
int i;
int s;

for (i=0; i < 200; i++)
      s = s + a1[i] + a2[i];
```

It is requested:

a) Reasonably indicate the number of lines and sets in the cache.
b) If the cache is considered to be empty and the values of variables i and s in the previous code are stored in records, indicate, considering only the vector accesses a1 and a2,the hit rate that is obtained when executing the previous code snippet.

Solution:

a) The cache has a size of 512 Kb = 2^{19} bytes. Each line has 128 bytes = 2^7 bytes. The number of lines is given by 2^{19} bytes / 2^7 bytes = 2^{12} lines = 4096 lines. As each set has 4 lines, the number of sets = 2^{12} / 2^2 = 2^{10} sets = 1024 sets

b) Since each element of the vector is a 4-byte integer, each line fits 128 / 4 = 32 integers. When `a1[0]` is first accessed, a miss occurs and a line that includes the elements is cached:
`a1[0], a1[1], ..., a1[31]`. The same goes for the vector a2. When elements
`a1[1], a2[1], ..., a1[31], a2[31]` are accessed, successes occur.
Every 32 accesses there is a failure. This process is repeated 6 times, until it reaches element 192. For the last 8 elements of the vector, there is one miss and 7 hits. Therefore, the number of accesses for each vector is 200 and the number of misses is 7, then the hit rate is 193 / 200 = 96.5%

Exercise 8. Be a 64-bit computer that has a cache of 512 KB, 4-ways associative and lines of 128 bytes. The cache access time is 2 ns and the miss penalty time is 120 ns. On this computer you want to run the following code snippet:

```
int m[32][32];      // matrix of integers of 32 x 32 elements.
                    // m elements are stored in rows.


for (i=0; i < 32; i = i + 1)
      for (j = 0; j < 32; j = j + 2)
            s = s + m[i][j];
```

It is requested:

a) Reasonably indicate the number of lines and sets in the cache.
b) If the cache is considered empty and the values of the variables i, j, and s in the preceding code are stored in registers, indicate, considering only the accesses to the array m, the hit rate that is obtained when executing the previous loop.
c) Calculate the total time spent accessing data in the previous loop.

Solution:

a) The cache has a size of 512 Kb = 2^{19} bytes. Each line has 128 bytes = 2^7 bytes. The number of lines is given by 2^{19} bytes / 2^7 bytes = 2^{12} lines = 4096 lines. As each set has 4 lines, the number of sets = 2^{12} / 2^2 = 2^{10} sets = 1024 sets

b) Since each element of the matrix is an 8-byte integer (since the computer is 64-bit), each line fits 128 / 8 = 16 integers. When `m[0][0]` is first accessed, a miss occurs and a line is cached that includes the elements: `m[0][0], m[0][1], ..., m[0][15]`. That is to say that every 8 accesses there is 1 failure. This process is repeated (32 x 16) / 8 = 64 times. Therefore, the total number of misses is 64 failures, and the number of successes is 512 – 64 = 448.

c) The access time is = 64 * 120 + 448 * 2 = 8576 ns.

Exercise 9. Consider the following code snippet:

```
for (i=0; i<64; i++) {
     for (j=0; j<1024; j=j+2 ){
          v[i][j] = v[i][j] * v[i][j+1] + b[i];
     }
}
```

This code is executed in an architecture with a word size of 8 bytes. The architecture consists of a cache size 256 KB, with block size of 64 bytes. The cache is fully associative and uses an LRU replacement algorithm. In the fragment of the previous code, ***v*** represents an array of 8-byte real numbers, which is stored by rows (C style) of size 64x1024; and ***b*** represents a vector of 64 real numbers of 8 bytes. It will be assumed that the index variables are stored in registers and that at the beginning of the code the cache is empty. Consider that in each iteration of the inner loop, the element `v[i][j]` is accessed first, and then the element `v[i][j+1]` is accessed.

It is requested:

a) Indicate the number of cache misses that occur when running the above code snippet.
b) The average miss rate.
c) If the average memory system access time is 6 ns and the cache access time is 4 ns, calculate the time required to handle a cache failure.

Solution:

a) The cache has a size of 256KB = 2^{18} bytes = 2^{12} lines

In each block of 64 bytes can be stored 64/8 = 8 elements of the matrix v and the vector b.

First, we will consider the accesses to the elements of v:

- Iteration i = 0;
  - Reading of v[0][0] Miss brings to cache v[0][0]… v[0][7]
  - Reading of v[0][1] Hit
  - Writing of v[0][0] Hit
  - Reading of v[0][2] Hit
  - Reading of v[0][3] Hit
  - Writing of v[0][2] Hit
  - Reading of v[0][4] Hit
  - Reading of v[0][5] Hit
  - Writing of v[0][4] Hit
  - Reading of v[0][6] Hit
  - Reading of v[0][7] Hit
  - Writing of v[0][6] Hit

- Reading of v[0][8] Miss brings to cache v[0][8]… v[0][15]
- Reading of v[0][9] Hit
- Reading of b[0] Hit
- Writing of v[0][8] Hit
- .... The pattern of accesses is repeated

In each iteration of the external loop, 512 * 3 = 1536 accesses to the v elements are made. Every 12 accesses there is a failure. In each iteration of the external loop there are, therefore, 1536/12 = 128 misses and 1408 hits to the elements of v. As there are 64 iterations of the external loop, there will be 64 * 128 = 8192 misses and 90112 hits on the accesses to the v array.

As for the elements of b, there is 1 miss every 8 iterations of the external dive. There are misses in the accesses to elements b[0], b[7], b[15]... Therefore there are 64 / 8 = 8 failures. The number of accesses to the elements of b is 64 * 512 = 32768, of which there are 8 are failures.
Therefore, the number of total accesses is: 64*512*4 = 131072 and the number of misses is 8192 + 8 = 8200.

b) The average miss rate is 8200 /131072=0.0625 => 6.25%
The average hit rate is 1 – 0.0625 = 0.9375 = > 93.75%

c) Average Time = h * Tc + (1-h) * Tf
h being the hit rate, Tc the cache time and Tf the time to deal with a miss.

6 = 0.9375 * 4 + 0.0625 * Tf
6 = 3.75 + 0.0625 * Tf
Tf = (6-3.75) / 0.0625 = 36 ns, is the time spent dealing with a miss.

Exercise 10. Be a 32-bit computer whose memory system like the one described below:

- A **DRAM-type main memory** of 512 Mbytes and an access time of 200 ns, including cache miss management.
- A **cache** with the following characteristics:
  - Size: 16 Kbytes
  - Line size: 64 bytes.
  - Correspondence function: 4-ways associative
  - Replacement Policy: FIFO (First In, First Out)
  - Access time: 10 ns.
  - The probability of cached success (h) is 95%

It is requested:

a) Describe the structure of the cache described, indicating the number of sets and the size of each set in bytes.
b) Calculate the average memory access time on that computer.
c) Indicate what format this cache will use to interpret memory addresses, describing what each part represents and its size in bits. The word size of the computer is 4 bytes.

Solution:

a) The cache has a size of 16 Kbytes = 16 x 2^{10} bytes = 2^{14} bytes

Number of cache entries or lines = 2^{14} bytes / 64 bytes (block size) = 256 cache lines

The number of sets = 256 / 4 = 64 sets (2^6 sets)

The size of the label field, the main memory has a size of 512 Mbytes = 512 x 2^{20} bytes, the number of blocks of main memory will be 2^{29} bytes / 64 bytes = 2^{23} blocks in main memory.

It is necessary to label 2^{23} blocks of main memory in 2^6 sets = > 2^{23} blocks / 2^6 sets = 2^{17}

Finally, if the computer has a word size of 4 bytes and there are blocks of 64 bytes, each block will fit 16 words (2^4)

b) The average memory access time in a single-level cache can be calculated with the following expression:

$$Tavg. = h * tc + (1-h) * tmp$$

Where:
- h = probability of a cached hit
- 1-h= probability of a cache miss
- tc = cache access time
- tmp= penalty time for miss.

So the average access time in this memory system would be:

$$Tavg. = 0.95 \times 10 + (1-0.95) \times 200 = 9.5 + 10 = 19.5 \text{ ns}$$

c) The format of a memory address will be as follows:
- Tag: 20 bits above the address.
- Assembly number: The following 6 bits.
- Byte inside the line: lower 6 bits.

Exercise 11. Be a computer equipped with a cache memory with the following characteristics:
A. Size: 16KB with 32-byte blocks (8 words)
B. Access time: 10ns

This memory is connected via a 32-bit bus to a main memory that has a latency time of 40 ns and can transfer 8 bytes every 10 ns. Calculate the hit rate that is necessary for the average access time to the memory system to be 20 ns.

Solution:

The cache line size is 32 bytes. The access time to a line is 40 ns + (32/8)*10 ns = 80 ns. To calculate the hit rate (h) the following expression can be used:

$$Tavg. = h * tc + (1-h) * Tmp$$

$20 = h * 10 + (1-h) * 80$ from which it is obtained that $h = 6/7 = 0.857$, that is, a hit rate of 85.7%.

Exercise 12. A 32-bit computer that addresses bytes has a 4 KB data cache and a 2 KB instruction cache. Both caches use a direct match function and their block size is 4 words. The access time to the memory system in case of success is 20 ns. In case of failure, the access time (including cache refresh) is 100 ns. The cache uses a deferred update policy (post-write, *write-back* or *copy-back).* The time to write a cache block to main memory is also 100 ns.

The following code snippet runs on the described computer, which is stored from the memory address 0x00F00000:

```
            mv   t0, zero
loop:       lw   t1, x(t0)
            sw   t1, y(t0)
            addi t0, t0, 4
            bne  t0, 16, loop
The end:
```

Where **x** corresponds to the memory address 0x70000000, and **y** corresponds to the memory address 0x00000400. All the instructions used in the code snippet take up one word each.

It is requested:

a) Determine the average memory access time for the described program.
b) If the data cache is replaced with another that uses a 4-input (4-way) associative matching function that uses a FIFO substitution policy, what will be the time?

Solution:

a) The instruction cache is 2 KB in size. Each entry in the cache has 4 words. Therefore:

- Block size = 4 words = 16 bytes.
- Number of blocks = 2KB / 16 bytes = 128 blocks

For the translation of addresses in the code cache, the following format will be used:

- 2 bits for byte selection within word.
- 2 bits for word selection within input.
- 7 bits for selecting one of the 128 inputs.
- 21 bits remaining as a tag.

In the proposed code fragment the first instruction is executed only once, and the rest of the instructions that make up the loop are executed 4 times. Instructions are stored from the memory address 0x00F00000. Therefore, the content of the memory in that area will be:

Memory address	Instruction
0x00F00000	**move $t 0, $zero**
0x00F00004	**lw $t 1, x($t 0)**
0x00F00008	**sw $t 1, y($t 0)**
0x00F0000C	**addi $t 0, $t 0, 4**
0x00F00010	**bne $t 0, 16, loop**

The addresses used are decomposed, therefore:

Address	Label	Input	Word	Byte
0x00F00000	0000 0000 1111 0000 0000 0	0000000	00	00
0x00F00004	0000 0000 1111 0000 0000 0	0000000	01	00
0x00F00008	0000 0000 1111 0000 0000 0	0000000	10	00
0x00F0000C	0000 0000 1111 0000 0000 0	0000000	11	00
0x00F00010	0000 0000 1111 0000 0000 0	0000001	00	00

It is noted that, in all references, the label is always the same. The sequence of references is:

Label	Input	Word	Hit/Miss
0x001700	0	0	Failure
0x001700	0	1	Success
0x001700	0	2	Success
0x001700	0	3	Success
0x001700	1	0	Failure
0x001700	0	1	Success
0x001700	0	2	Success
0x001700	0	3	Success
0x001700	1	0	Success
...	...	...	...
0x001700	0	1	Success
0x001700	0	2	Success
0x001700	0	3	Success
0x001700	1	0	Success

The total number of memory accesses for instructions is 17 (2 misses and 15 hits).

The data cache is 4 KB in size. Each entry in the cache has 4 words. Therefore:

- Block size = 4 words = 16 bytes.
- Number of blocks = 4 KB / 16 bytes = 256 blocks

For the translation of addresses in the data cache, the following format will be used:

- 2 bits for byte selection within word.
- 2 bits for word selection within input.
- 8 bits for selecting one of the 256 inputs.
- 20 bits remaining as label.

The code snippet performs 8 data accesses. The addresses of these accesses are:

Address	Label	Input	Word	Byte	Miss/Hit	Update
0x70000000	0x70000	0x00	00	00	Failure	No
0x00004000	0x00004	0x00	00	00	Failure	No
0x70000004	0x70000	0x00	01	00	Failure	Yes
0x00004004	0x00004	0x00	01	00	Failure	No
0x70000008	0x70000	0x00	10	00	Failure	Yes
0x00004008	0x00004	0x00	10	00	Failure	No
0x7000000C	0x70000	0x00	11	00	Failure	Yes
0x0000400C	0x00004	0x00	11	00	Failure	No

The total number of accesses is 8 (0 hits and 8 misses). In addition, 3 updates of the main memory occur.

If you take into account the access to instructions and data you have:

- Total number of accesses: 17 to instructions + 8 to data = 25 accesses
- Total number of hits: 15
- Total number of failures: 2 + 8 = 10
- Total number of updates: 3

The average access time will be:

$$T_{acceso} = \frac{15*20+10*100+3*100}{25} = \frac{300+1000+300}{25} = \frac{1600}{25} = 64$$

c) The instruction cache is the same, so the results of this part do not vary.

The data cache has sets of 4 entries. Each entry has 4 words

- Input size: 4 words = 16 bytes.
- Set size: 4 inputs = 16 words = 64 bytes.
- Number of sets = 4 KB / 64 bytes = 64 sets.

The format of a data address shall be:

- 2 bits for byte selection.
- 2 bits for word selection in input.
- 6-bit for assembly selection.
- 22 bits for the label.

The code snippet performs 20 data accesses. The addresses of these accesses are:

Address	Label	Group	Word	Byte	Miss/Hit	Update
0x70000000	0x70000.00	00 0000	00	00	Failure	No
0x00004000	0x00004.00	00 0000	00	00	Failure	No
0x70000004	0x70000.00	00 0000	01	00	Success	No
0x00004004	0x00004.00	00 0000	01	00	Success	No
0x70000008	0x70000.00	00 0000	10	00	Success	No

0x00004008	0x00004.00	00 0000	10	00	Success	No
0x7000000C	0x70000.00	00 0000	11	00	Success	No
0x0000400C	0x00004.00	00 0000	11	00	Success	No

Of the 8 accesses, 2 are misses and the remaining 6 are successes.

If you take into account the access to instructions and data you have:

- Total number of accesses: 17 to instructions + 8 to data = 25 accesses
- Total number of hits: 15 + 6 = 21
- Total number of failures: 2 + 2 = 4

The average access time will be:

$$T_{acceso} = \frac{21*20+4*100}{25} = \frac{420+400}{25} = \frac{820}{25} = 32{,}8$$

Exercise 13. A computer with the following characteristics is available:

- 64-bit word size.
- Main memory with an access time of 60ns.
- Internal data cache memory:
  - Capacity: 8KB.
  - 16-byte blocks.
  - Access time of 10ns.
  - Fully associative organization, with FIFO replacement policy (first in, first out)

Be the following program:

```
n = 0;
for (i = 0; i < 500; i++)
      n = n + b[i];
for (i = 0; i < 500; i++)
      a = a + b[i] * c[i];
for (i = 0; i < 500; i++)
      r = r + b[i] + d[i];
```

The loop control variable "I" and the variables "a", "r" and "n" are placed on processor registers. All variables are 64-bit integers.

Knowing that initially the data cache is empty. It is requested:

a) Calculate the cache miss rate generated by the program when running on the described computer.
b) Calculate the total time spent on data access.

Solution:

As each element of the vector is a 64-bit integer (8 bytes) and since the cache memory is 8KB, it has the capacity to store 8KB ÷ 8B per element = 1024 elements.

The number of elements of each of the vectors handled by the program is 500, so the cache memory has the capacity to locate two vectors and 24 additional integers.

On the other hand, since the block size is 16 bytes, each block accommodates 2 elements of a vector.

(a) Calculation of the miss rate

The first loop performs 500 accesses; the second loop performs 1000 accesses, and the third loop performs 1000 accesses. In **total, *2,500 accesses*** are made to the elements of vectors b, c and d.

In the first loop there are 250 misses:

i = 0 : b[0] is not in the cache, miss. B[0] and b[1] are taken to block 0 of the cache
i = 1: b[1] is in the cache, hit.
i = 2: b[2] is not in the cache, miss. B[2] and b[3] are taken to block 1 of the cache
i = 3: b[3] is in the cache, hit.
...
Fails every 2 iterations (500 ÷ 2 = 250)

In the second loop, although the vector b is referenced again, the elements of b are in the cache, therefore, **no** misses occur due to the use of b. One miss occurs for every two elements of c, in total 250 misses occur.

When the third loop begins, there is only room for 24 new values of c, when the element c[24] is required, a victim must be chosen to leave the cache, as the policy chosen is FIFO that victim corresponds to the block that has the values b[0] and b[1], which fortunately have already been used and are not needed for subsequent calculations. The effect will continue to cascade and therefore, the same number of misses will occur as in the previous loop, this is 250 failures.

The total number **of misses** is 250 + 250 + 250 = **750. Miss rate** is 100 x (750 misses ÷ 2,500 accesses) = **30%**

b) Total time spent on data access:

$T = TotalHits \times T_{acceso\_cache} + TotalMiss \times (T_{acceso\_cache} + T_{acceso\_memoria})$

$= 1{,}750 \times 10ns + 750 \times (10ns + 60ns) =$ **70,000ns = 70µs = 0.07 ms**

Exercise 14. It has a computer that directs the memory by bytes and has a cache with a size of 32 KB to save instructions or data of the processes. The size of the line is 64 bytes. Processes can address 1MB of main memory and this is addressed by bytes. A RAM access consumes 80 ns and the cache 30 ns. Assume that the cache is initially empty

There are two design alternatives for the cache:
1. Direct correspondence.
2. Associative correspondence by 8-way sets with LRU replacement algorithm.

For each of these two design alternatives indicate:
a) Given the following sequence of addresses:
0x7B042 0x7D042 0x7D074
identify the tag, line (or set), and offset in which the data associated with each address is located, and which of these addresses cause a cache failure.
b) The access time of the sequence.

Solution:

First, the case of direct correspondence will be analyzed. The cache size is 32KB (2^{15} bytes), the line is 64B (2^6 bytes), the number of cache entries is: 2^{15} bytes ÷ 2^6 bytes = 2^9 inputs.
The number of bits to save data or instructions: 64B x 8 = 2^6 x 2^3 = 2^9 bits.
Bits for the label field. Main Memory Size ÷ Cache Size = 1MB ÷ 32KB = 2^{20} ÷ 2^{15} = 2^5, i.e. 5 bits.

That is, the top 5 bits will be used for the label, the intermediate 9 for the line and the bottom 6 bits for indicating the byte within the line.

For the sequence: 0x7B042 0x7D042 0x7D074 you get the following:

7B042 is: 01111 011000001 000010 – miss, empty cache-
7D042 is: 01111 101000001 000010 – miss, different lines–
7D074 is: 01111 101000001 110100 – hit with 0x0007D042 same label, same line-

A hit is done in 30ns and a miss in 30ns + 80ns = 110ns. The access time is given by 2 x 110ns + 1 x 30ns = 250ns

Next, the case of associative correspondence by sets of 8 ways is analyzed. The cache size is 32KB (2^{15} bytes) and 8 ways, the line is 64B (2 6 bytes), the number of cache entries is: 2 15 bytes ÷ (2^6 bytes x $2^{3)}$=2^6 sets of 8 lines each. The bottom 6 bits are used to identify the byte. The next 6 bits to identify the set and the top 20-6-6 = 8 bits for the tag.

For the sequence: 0x7B042 0x7D042 0x7D074 you get:

7B042 is: 01111011 000001 000010 – miss, empty cache-
7D042 is: 01111101 000001 000010 – miss
7D074 is: 01111101 000001 110100 – hit, the same set that 0x7D042 and matches the 0x7D042 so it is the same block of memory–

A hit is done in 30ns and a miss in 30ns + 80ns = 110ns. The access time is given by 2 x 110ns + 1 x 30ns = 250ns

Exercise 15. Be a 32-bit computer that has a 64 KB instruction cache and a 128 KB data cache. Both memories have 32-byte lines, direct correspondence policy, and deferred or delayed write policy. Memory is addressed by bytes. Consider the following code snippet:

```
for (i = 0; i< 512; i++)
        for (j = 0; j <  512; j++)
                c[i][j] = a[i][j] + b[i][j]
```

Where "a", "b", and "c" represent square matrices of 512 x 512 integers, which are stored in memory consecutively. Each array in turn is stored in rows. The matrix "a" begins in the hexadecimal direction 0x00100000. It is requested:

a) Calculate the hit rate that occurs in the data cache for the previous code snippet, assuming that the data cache is initially empty.
b) Propose a cache layout that improves the hit rate for the previous program fragment without an excessive increase in cache search time. Calculate the hit rate for this design.

Solution:

a) The data cache has 128 KB / 32 bytes = $2^{17}/2^5 = 2^{12} = 4096$ lines or blocks of 32 bytes. Each array consists of 512 x 512 x 4 = 1 MB = 2^{20} bytes. The three vectors are stored consecutively, first "a", then "b", and then "c". The starting directions of the three vectors are as follows:

- Matrix a begins at position 0x00100000 and ends at position 0x001FFFFF
- Matrix b begins at position 0x00200000 and ends at position 0x002FFFFF
- Matrix c begins at position 0x00300000 and ends at position 0x003FFFFF

To determine on which cache line a memory address is stored, you must use the following distribution of the bits in an address:

- The top 15 bits constitute the label of the line.
- The following 12 bits determine the block number and therefore the cache line in which the block must be stored.
- The bottom 5 bits determine the byte within the line.

In the above program fragment there are 512 * 512 * 3 = 786432 memory accesses, of which 2/3 are reads (arrays "a" and "b") and 1/3 are writes (the "c" matrix). The following are the accesses that occur to the cache:

Element	Memory address	Type of access	Cache line	Miss/hit
a[0][0]	0x00100000	Reading	0	Miss
b[0][0]	0x00200000	Reading	0	Miss (expelled a)
c[0][0]	0x00300000	Writing	0	Miss (expelled b)
a[0][1]	0x00100004	Reading	0	Miss (ejected c)
b[0][1]	0x00200004	Reading	0	Miss (ejected a)
c[0][1]	0x00300004	Writing	0	Miss (expelled b)

This process is repeated until the end, that is, all accesses are failures. The hit rate is therefore 0.

b) The fact that all cache accesses are misses is due to the fact that at each turn of the loop the elements a[i][j], b[i][j] and c[i][j] are stored in the same cache line. A 4-way set associative cache could be used to resolve this issue. In this way, at each turn of the loop, three blocks corresponding to vectors a, b and c could be stored in the same set of cache. As each line fits 8 elements, there would be one miss for every 8 accesses, which would obtain a hit rate of 7/8.

Exercise 16. Be a 32-bit computer that addresses memory by bytes and has a cache for instructions and a cache for data. Both memories have 16-byte lines and an associative match policy by sets of n-way with LRU replacement algorithm.

Consider the following code snippet:

```
for (i = 0; i< 218; i++)
      a[i] = b[i] + c[i] + d[i];
```

Where a, b, c and d represent vectors of 2^{18} integers, which are stored in memory consecutively. The vector "a" begins in the hexadecimal direction 0x00500000. Next, "b", "c" and finally "d" are stored. Whereas the cache is initially empty. It is requested:

a) For a data cache size of 256KB, two degrees of associativity are shuffled: **n=2 or n=4.** Reasonably calculate what the hit rate is in each of the two cases and compare both scenarios.
b) Considering the above two scenarios, would doubling the cache size to 512KB increase the hit rate?

Solution

a) Each vector consists of 2^{18} x 4 bytes (each integer) = 2^{20} bytes. All four vectors are stored consecutively. The starting directions of the three vectors are as follows:

- Vector "a" begins at position 0x00500000 and ends at position 0x005FFFFF
- Vector "b" begins at position 0x00600000 and ends at position 0x006FFFFF
- Vector "c" begins at position 0x00700000 and ends at position 0x007FFFFF
- Vector "c" begins at position 0x00800000 and ends at position 0x008FFFFF

Degree of associativity n=2. The data cache is divided into 2-line sets. The size of the assembly is therefore 2 x 16 bytes = 32 bytes = 2^5 bytes. The number of sets in the caches is 256 KB / 32 bytes = 2^{18} / 2^5 = 2^{13} sets.

To determine in which cache set a memory address is stored, you must use the following distribution of the bits in an address:

- The top 15 bits constitute the line label
- The following 13 bits determine the set in which the block is stored. Within this set the block will be stored on either line following an LRU replacement policy.
- The bottom 4 bits determine the byte within the line.

The following are the accesses that occur to the cache:

Element	Memory address	Type of access	Cache set	Line within the set	Miss/Hit
b[0]	0x00600000	Reading	0	0	Miss
c[0]	0x00700000	Reading	0	1	Miss
d[0]	0x00800000	Reading	0	0	Miss (expelled b)
a[0]	0x00500000	Writing	0	1	Miss (ejected c)
b[1]	0x00600004	Reading	0	0	Miss (expelled d)
c[1]	0x00700004	Reading	0	1	Miss (expelled a)
d[1]	0x00800004	Reading	0	0	Miss (expelled b)
a[1]	0x00500004	Writing	0	1	Miss (ejected c)
...	...	...	...	...	...

b[4]	0x00600010	Reading	1	0	Miss
c[4]	0x00700010	Reading	1	1	Miss
d[4]	0x00800010	Reading	1	0	Miss (expelled b)
a[4]	0x00500010	Writing	1	1	Miss (ejected c)

This process is repeated until the end, that is, all accesses are failures. The hit rate is therefore 0.

Degree of associativity n=4: The data cache is divided into sets of four lines. In this case the number of sets is 256 KB / 64 bytes = 2^{18} / 2^{6} = 2^{12} sets

To determine in which cache set a memory address is stored, you must use the following distribution of the bits in an address:

- the top 16 bits constitute the line label
- The following 12 bits determine the set in which the block is stored. Within this set the block will be stored on either line following an LRU replacement policy.
- The bottom 4 bits determine the byte within the line.

The following are the accesses that occur to the cache:

Element	Memory address	Type of access	Cache set	Line within the set	Miss/Hit
b[0]	0x00600000	Reading	0	0	Miss
c[0]	0x00700000	Reading	0	1	Miss
d[0]	0x00800000	Reading	0	2	Miss
a[0]	0x00500000	Writing	0	3	Miss
b[1]	0x00600004	Reading	0	0	Hit
c[1]	0x00700004	Reading	0	1	Hit
d[1]	0x00800004	Reading	0	2	Hit
a[1]	0x00500004	Writing	0	3	Hit
b[2]	0x00600008	Reading	0	0	Hit
c[2]	0x00700008	Reading	0	1	Hit
d[2]	0x00800008	Reading	0	2	Hit
a[3]	0x00500008	Writing	0	3	Hit
b[3]	0x0060000C	Reading	0	0	Hit
c[3]	0x0070000C	Reading	0	1	Hit
d[3]	0x0080000C	Reading	0	2	Hit
a[3]	0x0050000C	Writing	0	3	Hit
b[4]	0x00600010	Reading	1	0	Miss
c[4]	0x00700010	Reading	1	1	Miss
d[4]	0x00800010	Reading	1	2	Miss
a[4]	0x0050000C	Writing	1	3	Miss
...	...	...	...	...	...

This process is repeated until the entire cache is filled and also until the vectors have just been processed. Thus, for each set of the cache there will be 4 misses and 12 hits, so the hit rate will be (12/16)x100 = 75%.

b) Increasing the cache size does not reduce the hit rate since there is no temporary locality in data access.

Exercise 17. Given the following code that runs on a 32-bit architecture, where array A is an array of integers (size 4 bytes) that is stored in rows. That is, the entries are stored in memory according to the following order: A[1][1], A[1][2], A[1][3].... A[1][1024], A[2][1], ... The matrix has the following dimensions: A[1024][8]. Given the following loop:

```
for(j=0;j<8;j++)
      for(i=0;i<1024;i++)
            A[i][j]=8
```

The architecture implements a cache with direct correspondence. This memory has a capacity of 16KB and a line size of 32 bytes. Variables i and j are stored in logs.

It is requested:

a) Describes the behavior of the program on the data cache.
b) Calculate the hit rate in the cache.
c) If the cache access time is 5 clock cycles and the memory access time is 15 cycles, calculate the average time (in cycles) of loop execution.
d) Justifiably indicates which transformations can be performed on the code in order to improve its performance.

Solution:

Because the matrix A is accessed by columns there will be a small spatial locality in the accesses. Of each line loaded into the cache, only one word will be read (because the next word is 8 words away, that is, in the next cache line.).

a) When the program runs, one word from each line is accessed, accessing consecutive cache lines. The cache accommodates 16KB/32=4K lines, while the array has a size of 1024*8*4=32Kb=8K lines. For this reason, when a line is accessed again (after performing 1024 loop i accesses) the memory no longer contains it, causing a cache failure.
b) The cache hit rate is 0%, because a piece of data is never reused in this memory.
c) Because the hit rate is 0%, all accesses are made to memory, so the total execution time (in cycles) will be: (Mem_cycles + Cache_cycles)*Num_access = (15 + 5)*8K.
d) By exchanging the indices of the loop it is possible to maximize the spatial locality in the accesses. Now the complete content of the cache line is reused, so the hit rate will be 7 hits for each miss and the total execution time, in cycles, will be (5*7+*(15+5)*1)*1K.