Chapter 7

Memory Hierarchy

7.1 Storage Technologies

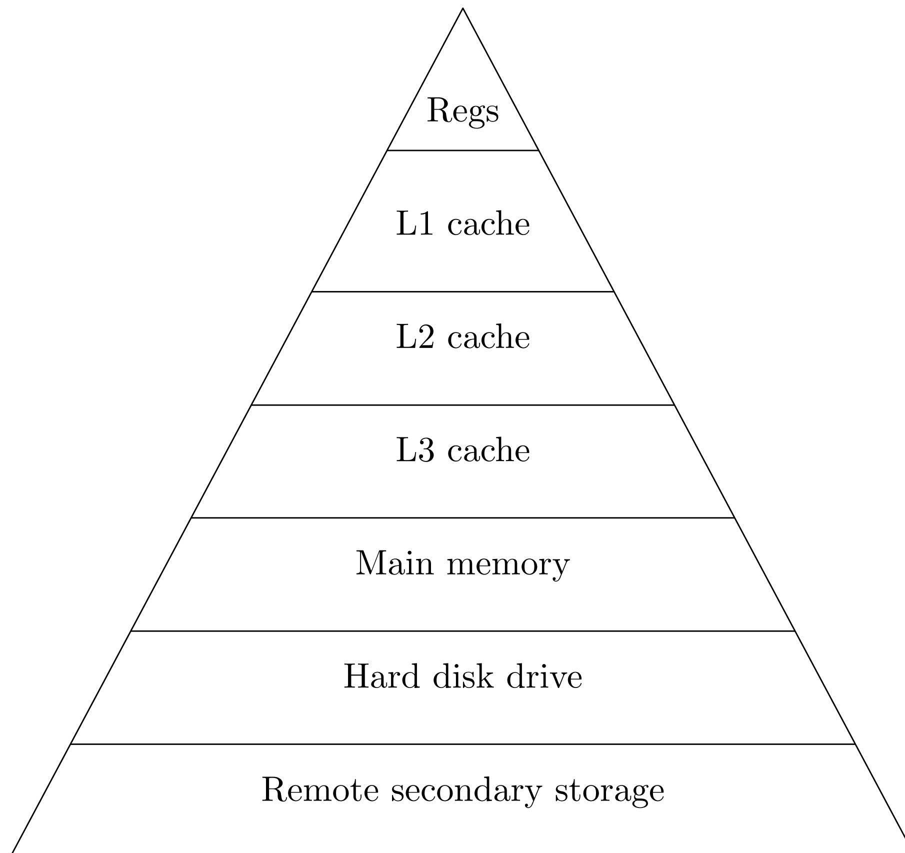
1. Random-Access Memory
   (a) DRAM (Dynamic Random Access Memory)
   (b) SRAM (Static Random Access Memory)
2. Magnetic disk – secondary storage
3. Solid-state drive – secondary storage

7.1.1 SRAM versus DRAM

Characteristic	SRAM	DRAM
Transistors per bit	6	1
Relative access time	1×	10×
Persistent?	Yes	No
Sensitive?	No	Yes
Relative cost	100×	1×

7.2 Memory Hierarchy

Different types of memories with different storage technologies are used together in a computer system. This is a trade-off between memory access time and cost.



7.3 Cache Memory

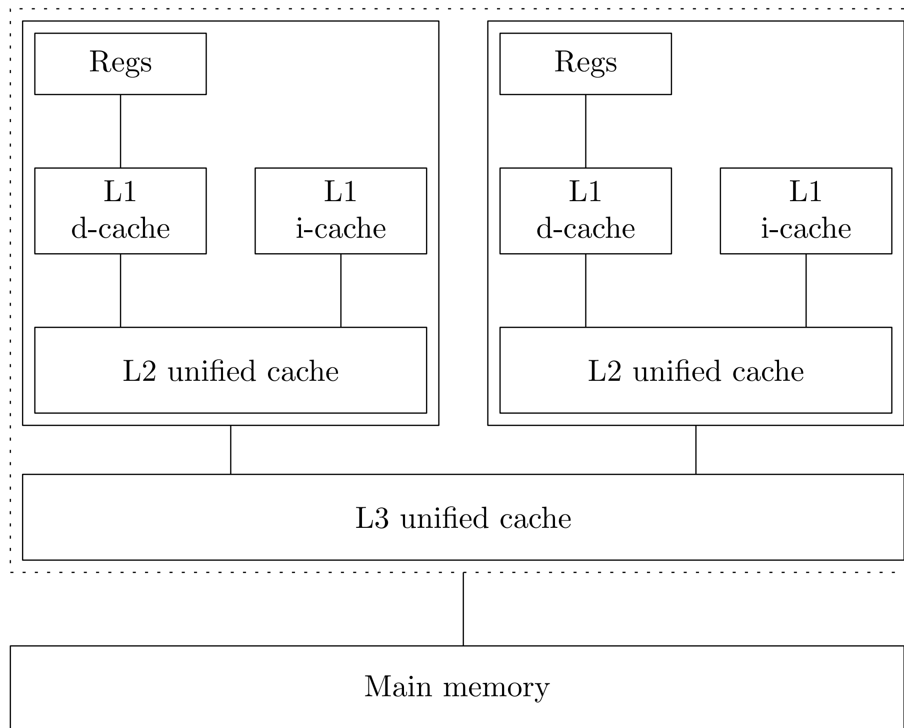
Instead of reading/writing a piece of data (e.g. 4 bytes of integer) from/to the main memory (DRAM) every time, the processor reads a block of data (e.g. 64 bytes) from the main memory and put the block into L3/L2/L1 cache memories (SRAM).

When the processor continues accessing the consecutive piece of data in the same block, the data can be retrieved from the cache which is faster than the main memory. This therefore improves the data access speed.

However, the size of cache memory is relatively small due to its cost. All the necessary data cannot be kept in the cache. When the cache is full and the processor is requesting to store a new block. The *least-recently used* block in the cache will be replaced by the new block.

More cache memory space means *lower* chance of a block replacement. The performance is then better.

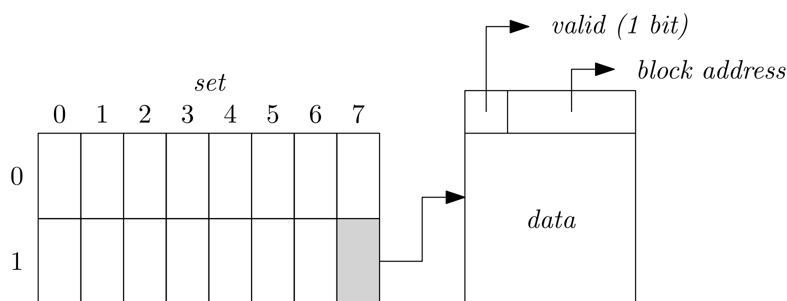
"*Cache hit*" is when the data can be accessed from the cache. "*Cache miss*" is when the data is only in the main memory. The block containing that data will be loaded to the cache.


Regs
L1
d-cache
L1
i-cache
L2 unified cache
Regs
L1
d-cache
L1
i-cache
L2 unified cache
L3 unified cache
Main memory

7.4 Set Associative Cache

An n-way set associative cache is the most widely used structure of cache memories. The cache composes m sets. Each set can stored n blocks of data. Each block contains b bytes of data.

Example 7.1 a 2-way set associative cache with block size of 64 bytes and 8 sets.



Since the cache size is limited, each block contains a 1-bit space called *valid* to indicate if the block is storing a block of data from the main memory. The block is storing data when *valid* = 1. The *block address* specifies where the data came from, or where the data is originally located in the main memory.

The main memory is also separated into blocks of the same size. Each block has its own address used with the cache. For example,

Memory address	Block Address (block size = 16 bytes)
00 - 15	0
16 - 31	1
32 - 47	2
48 - 63	3
...	...

$$B = \left\lfloor \frac{M}{b} \right\rfloor \tag{7.1}$$

where B is the block address, M is the memory address, and b is the block size.

In the set associative cache, blocks cannot be freely stored. Each block has its designated set number calculated from the block address. This is to simplify the logic to control the cache memory as well as ensuring the distribution of data.

Given a block address B, its designated set number is

$$s = B \bmod m \tag{7.2}$$

where s is the set number and m is the number of sets in the cache.

For example,

Block Address	Set Number (8 sets)
0	0 mod 8 = 0
5	5 mod 8 = 5
13	13 mod 8 = 5
16	16 mod 8 = 0

Example 7.2 Given a 2-way set associative cache with block size of 64 bytes and 8 sets. Show how the data are stored in the cache when the processor requests the following block addresses.

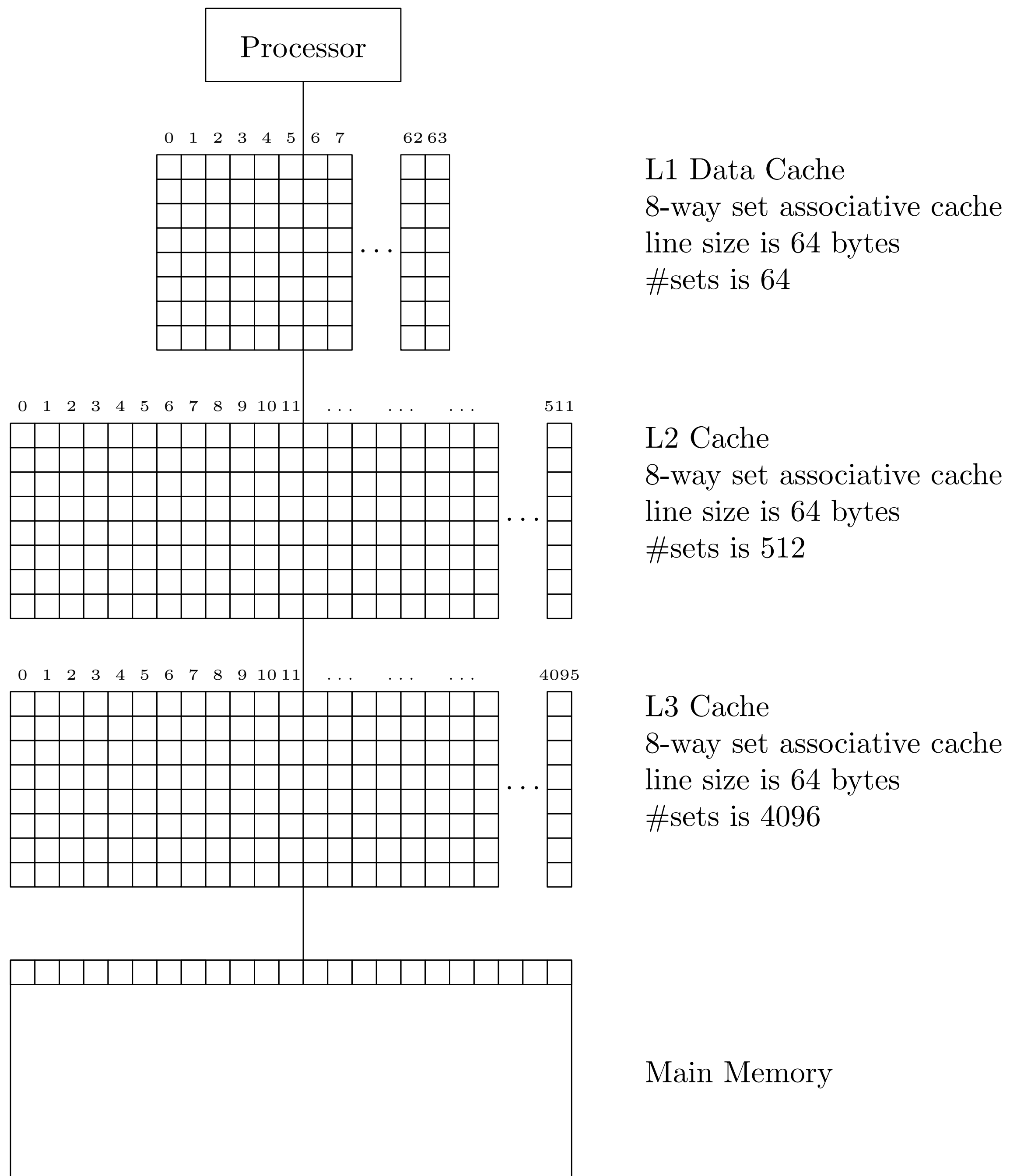
$$0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 0 \rightarrow 8 \rightarrow 16 \rightarrow 24 \rightarrow 32$$

Exercise 7.1 Given a 2-way set associative cache with block size of 64 byte and 16 sets, show how the data are stored in the cache when the processor requests the following block addresses.

$$0 \rightarrow 1 \rightarrow 3 \rightarrow 16 \rightarrow 15 \rightarrow 48 \rightarrow 49 \rightarrow 63 \rightarrow 127$$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

7.5 Multi-Level Cache



Exercise 7.2 When the processor reads a 4-byte integer from the following address, where is the integer stored in the L1, L2, and L3 caches shown in the previous page? Assume that all the caches are empty.

1. Address `40000`

2. Address `204355`

7.6 Locality and Cache-Friendly Code

The memory performance depends on how the order of the memory addresses accessed by the processor. We can control the order by carefully write the programs.

Exercise 7.3

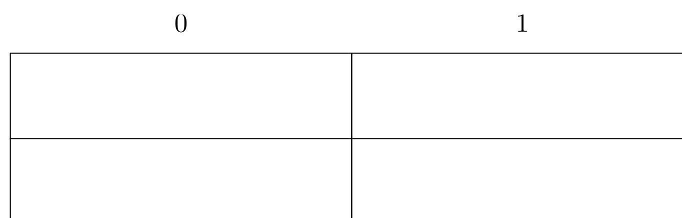
```
#define N 8

int sum_1d_array(int a[N]) {
  int i;
  int sum=0;

  for(i=0; i<N; i++) {
    sum += a[i];
  }

  return sum;
}
```

Show how the elements are accessed if we have 2-way set associative cache with 2 sets, and cache line size is 16 bytes (4 integers). Here, we assume that the address of the array is `0`.



Write hits or misses in the following table.

$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$	$i = 6$	$i = 7$

Exercise 7.4

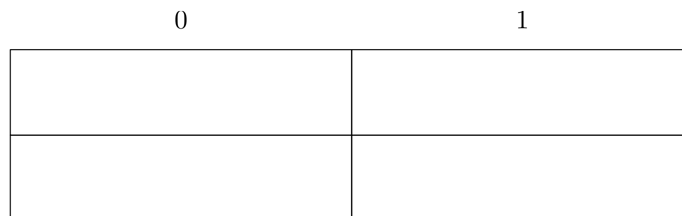
```
#define ROWS 4
#define COLS 8

int sum_2d_array_rows(int a[ROWS][COLS]) {
  int i, j;
  int sum=0;

  for(i=0; i<ROWS, i++) {
    for(j=0; j<COLS; j++) {
      sum += a[i][j];
    }
  }

  return sum;
}
```

Show how the elements are accessed if we have 2-way set associative cache with 2 sets, and cache line size is 16 bytes (4 integers). Here, we assume that the address of the array is `0`.



Write hits or misses in the following table.

	$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$	$j=7$
$i=0$								
$i=1$								
$i=2$								
$i=3$								

Exercise 7.5

```
#define ROWS 4
#define COLS 8

int sum_2d_array_cols(int a[ROWS][COLS]) {
  int i, j;
  int sum=0;

  for(j=0; j<COLS, j++) {
    for(i=0; i<ROWS; i++) {
      sum += a[i][j];
    }
  }

  return sum;
}
```

Show how the elements are accessed if we have 2-way set associative cache with 2 sets, and cache line size is 16 bytes (4 integers). Here, we assume that the address of the array is `0`.

Write hits or misses in the following table.

	$j = 0$	$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$	$j = 6$	$j = 7$
$i = 0$								
$i = 1$								
$i = 2$								
$i = 3$								