# CS 449 - Tutorial on Caches

## IEC Prefixing System

We often need to express large numbers and the preferred tool for doing so is the IEC Prefixing System!

| Kibi- | (Ki) | $2^{10} \approx 10^3$ | Pebi- | (Pi) | $2^{50} \approx 10^{15}$ |
|---|---|---|---|---|---|
| Mebi- | (Mi) | $2^{20} \approx 10^6$ | Exbi- | (Ei) | $2^{60} \approx 10^{18}$ |
| Gibi- | (Gi) | $2^{30} \approx 10^9$ | Zebi- | (Zi) | $2^{70} \approx 10^{21}$ |
| Tebi- | (Ti) | $2^{40} \approx 10^{12}$ | Yobi- | (Yi) | $2^{80} \approx 10^{24}$ |

## Prefix Exercises:

Write the following as powers of 2.  The first one has been done for you:

| | | |
|---|---|---|
| 2 Ki-bytes = $\mathbf{2^{11}}$ **bytes** | 64 Gi-bits = | 16 Mi-integers = |
| 256 Pi-pencils = | 512 Ki-books = | 128 Ei-students = |

Write the following using IEC Prefixes.  The first one has been done for you:

| | | |
|---|---|---|
| $2^{15}$ cats = **32 Ki-cats** | $2^{34}$ birds = | $2^{43}$ huskies = |
| $2^{61}$ things = | $2^{27}$ caches = | $2^{58}$ addresses = |

---

## Accessing a Cache (Hit or Miss?)

Assume the following caches all have block size $K = 4$ and are in the current state shown (you can ignore "−").
All values are shown in hex.  Tag fields are NOT padded, while bytes of the cache blocks are shown in full. The word size for the machine with these caches is 12 bits (i.e. addresses are 12 bits long)

### Direct-Mapped:

| Set | Valid | Tag | B0 | B1 | B2 | B3 |
|---|---|---|---|---|---|---|
| 0 | 1 | 15 | 63 | B4 | C1 | A4 |
| 1 | 0 | — | — | — | — | — |
| 2 | 0 | — | — | — | — | — |
| 3 | 1 | D | DE | AF | BA | DE |
| 4 | 0 | — | — | — | — | — |
| 5 | 0 | — | — | — | — | — |
| 6 | 1 | 13 | 31 | 14 | 15 | 93 |
| 7 | 0 | — | — | — | — | — |

| Set | Valid | Tag | B0 | B1 | B2 | B3 |
|---|---|---|---|---|---|---|
| 8 | 0 | — | — | — | — | — |
| 9 | 1 | 0 | 01 | 12 | 23 | 34 |
| A | 1 | 1 | 98 | 89 | CB | BC |
| B | 0 | 1E | 4B | 33 | 10 | 54 |
| C | 0 | — | — | — | — | — |
| D | 1 | 11 | C0 | 04 | 39 | AA |
| E | 0 | — | — | — | — | — |
| F | 1 | F | FF | 6F | 30 | 0 |

Offset bits: _____

Index bits: _____

Tag bits: _____

| | Hit or Miss? | Data returned |
|---|---|---|
| a)  Read 1 byte at `0x7AC` | | |
| b)  Read 1 byte at `0x024` | | |
| c)  Read 1 byte at `0x99F` | | |

<u>2-way Set Associative:</u>

| Set | Valid | Tag | B0 | B1 | B2 | B3 |
|---|---|---|---|---|---|---|
| 0 | 0 | — | — | — | — | — |
| 1 | 0 | — | — | — | — | — |
| 2 | 1 | 3 | 4F | D4 | A1 | 3B |
| 3 | 0 | — | — | — | — | — |
| 4 | 0 | 6 | CA | FE | F0 | 0D |
| 5 | 1 | 21 | DE | AD | BE | EF |
| 6 | 0 | — | — | — | — | — |
| 7 | 1 | 11 | 00 | 12 | 51 | 55 |

| Set | Valid | Tag | B0 | B1 | B2 | B3 |
|---|---|---|---|---|---|---|
| 0 | 0 | — | — | — | — | — |
| 1 | 1 | 2F | 01 | 20 | 40 | 03 |
| 2 | 1 | 0E | 99 | 09 | 87 | 56 |
| 3 | 0 | — | — | — | — | — |
| 4 | 0 | — | — | — | — | — |
| 5 | 0 | — | — | — | — | — |
| 6 | 1 | 37 | 22 | B6 | DB | AA |
| 7 | 0 | — | — | — | — | — |

Offset bits: _____

Index bits: _____

Tag bits: _____

| | Hit or Miss? | Data returned |
|---|---|---|
| a) Read 1 byte at `0x435` | | |
| b) Read 1 byte at `0x388` | | |
| c) Read 1 byte at `0x0D3` | | |

<u>Fully Associative:</u>

| Set | Valid | Tag | B0 | B1 | B2 | B3 |
|---|---|---|---|---|---|---|
| 0 | 1 | 1F4 | 00 | 01 | 02 | 03 |
| 0 | 0 | — | — | — | — | — |
| 0 | 1 | 100 | F4 | 4D | EE | 11 |
| 0 | 1 | 77 | 12 | 23 | 34 | 45 |
| 0 | 0 | — | — | — | — | — |
| 0 | 1 | 101 | DA | 14 | EE | 22 |
| 0 | 0 | — | — | — | — | — |
| 0 | 1 | 16 | 90 | 32 | AC | 24 |

| Set | Valid | Tag | B0 | B1 | B2 | B3 |
|---|---|---|---|---|---|---|
| 0 | 0 | — | — | — | — | — |
| 0 | 1 | AB | 02 | 30 | 44 | 67 |
| 0 | 1 | 34 | FD | EC | BA | 23 |
| 0 | 0 | — | — | — | — | — |
| 0 | 1 | 1C6 | 00 | 11 | 22 | 33 |
| 0 | 1 | 45 | 67 | 78 | 89 | 9A |
| 0 | 1 | 1 | 70 | 00 | 44 | A6 |
| 0 | 0 | — | — | — | — | — |

Offset bits: _____

Index bits: _____

Tag bits: _____

| | Hit or Miss? | Data returned |
|---|---|---|
| a) Read 1 byte at `0x1DD` | | |
| b) Read 1 byte at `0x719` | | |
| c) Read 1 byte at `0x2AA` | | |

## Code Analysis

Consider the following code that accesses a <u>two-dimensional</u> array (of size 64×64 `int`s).
Assume we are using a direct-mapped, 1 KiB cache with 16 B block size.

```
for (int i = 0; i < 64; i++)
    for (int j = 0; j < 64; j++)
        array[i][j] = 0;          // assume &array = 0x600000
```

a) What is the miss rate of the execution of the entire loop?

b) What code modifications can <u>change</u> the miss rate?  Brainstorm before trying to analyze.

c) What cache parameter changes (size, associativity, block size) can <u>change</u> the miss rate?

## Cache Simulator Demo

Let's get some practice with the cache simulator! First, go to:

`https://www.cs.pitt.edu/~vinicius/cachesim/`

At the top you'll see 4 boxed regions:

- <u>System Parameters</u> [†]     This lets you play around with the structure/format of the cache
- <u>Manual Memory Access</u> [†]   This is where you actually make reads and writes to memory
- <u>History</u>                    An interactive log of executed accesses. You can type/paste accesses here, too!
- <u>Simulation Messages</u>        Describes the most recent actions made by the simulator.

[†] These include "Explain" toggles that walk you through execution step-by-step.

---

**a)** Set the following System Parameters (but *don't* generate the system yet):

<u>Address Width</u> → **6**, <u>Cache Size</u> → **16**, <u>Block Size</u> → **4**, <u>Associativity</u> → **2**, leave the rest at default values.

Based on just the system parameter numbers above shown, predict the following:

   i) Highest memory address: 0b_____        ii) Number of sets in cache: _____

[*Click "Generate System" to verify your responses* ]

**b)** We are about to **READ** the byte at the address **0x2A**. Predict the following:

   i) This block will be placed in set #: _____        ii) The stored tag bits will be: 0b_____

   iii) The 4 bytes of *data* in this block are (in order):   0x_____, 0x_____, 0x_____, 0x_____

[*Enter "2a" into the Read Addr and click "Read" to verify your responses*]

**c)** We are about to **WRITE** the byte **0xB1** to the address **0x1B**. Predict the following:

   i) This block will be placed in set #: _____        ii) The stored tag bits will be: 0b_____

[*Enter "1b" into the Write Addr and "b1" into the Write Byte and then click "Write" to verify your responses* ]

   iii) Notice that the value of the byte at address 0x1B is different in the cache and memory.

   What indicates this disparity in the cache? _____

   What would have happened if our write miss policy were "**No Write-Allocate**" instead?

**d)** We are about to **READ** the byte at address **0x01**. Predict the following:

   i) This block will be placed in set #: _____        ii) The stored tag bits will be: 0b_____

   iii) Will this access cause a conflict/replacement? (circle one)          Yes               No

   iv) If yes, which block will be evicted? (circle one)          Read from (b)        Write from (c)

[*Enter "01" into the Read Addr and click "Read" to verify your responses* ]

**e)** We are about to **WRITE** the byte **0xE9** to the address **0x1C**. Predict the following:

   i) This block will be placed in set #: _____        ii) The stored tag bits will be: 0b_____

   iii) Will this access cause a conflict/replacement? (circle one)  Yes     No

   iv) If yes, which block will be evicted?          Read from (b)        Write from (c)        Read from (d)

[*Enter "1c" into the Write Addr and "e9" into the Write Byte and then click "Write" to verify your responses* ]

**f)** At this point, your **History** should show:

```
R(0x2a)  = M
W(0x1b,  0xb1)  = M
R(0x01)  = M
W(0x1c,  0xe9)  = M
>
```

*Append* the bolded text below so that your History looks like:

```
R(0x2a)  = M
W(0x1b,  0xb1)  = M
R(0x01)  = M
W(0x1c,  0xe9)  = M
> W(0x03,  0xff)
R(0x27)
R(0x10)
W(0x1d,  0x00)
```

[*Click "Load." You'll notice that " = ?" is appended to each of these new memory accesses*]

Predict if '?' will resolve to Hit (H) or Miss (M) for each of the new accesses:
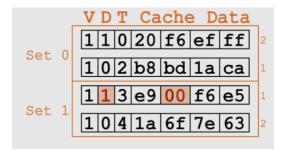
    i) W(0x03, 0xff) = _____     ii) R(0x27) = _____

    iii) R(0x10) = _____     iv) W(0x1d, 0x00) = _____

[*Click the down arrow (↓) to verify your responses for each access*]

**g)** The cache, after the 8 executions detailed above, should look like this:



```
            V D T  Cache  Data
           1 1 0 20 f6 ef ff   2
  Set 0
           1 0 2 b8 bd 1a ca   1
           1 1 3 e9 00 f6 e5   1
  Set 1
           1 0 4 1a 6f 7e 63   2
```

The small numbers on the right (outside of the sets) indicate how recently used each line is within the set, with smaller numbers being *more recently* used).

    i) An **LRU** replacement policy will evict which block on the next conflict in set 0?　　Line 1　　Line 2

    ii) What is one benefit of using **LRU** over **Random**?

    iii) What is one benefit of using **Random** over **LRU**?

**h)** If we were to flush the cache right now how many bytes in memory would change?　　_____

How many bytes would change if we were using **Write Through** instead of **Write Back**?　　_____

Can you explain why these numbers are the same/different? (if not, try changing the write hit policy and re-running using the history above).