**Why do we cache? What is the end result of our caching, in terms of capability?**

We cache as a tradeoff between space and speed. Fast memory is expensive and cheap memory is slow. Caching give us extra layers of fast memory between our program and the largest memory layer. Caching gets us the speed of the fast memory with the spatial capacity of the largest memory.

**What are temporal and spatial locality? Give high level examples in software of when these occur.**

Temporal locality – a recently-accessed item will often be accessed again soon. (i.e. a commonly-executed piece of code, such as a menu or library function)

Spatial locality – we tend to access nearby items to ones that have been accessed recently. (i.e. structs or sequential array accesses)

**Define the following cache terms:**

Cache hit – requested address already in the cache (fast return)

Cache miss – cache index "empty" (not valid), so read from memory and write to cache (slow).

Cache miss, block replacement – cache index has wrong block (non-matching tag), so read from memory and override block (slow).

## T:I:O Problems

**TIO breakdown of an address:**

| **T**ag | **I**ndex | **O**ffset |
|---|---|---|

**Offset**: "column index" (O bits) – location of address within block
**Index**: "row index" (I bits) – location (row) of block within cache
**Tag**: "block identifier" (T bits) – is this the right block?

Fill out the following table. Assume all caches are write-through.

| Address Bits | Cache Size (Data) | Cache Type | Block Size | Tag Bits | Index Bits | Offset Bits | Bits per Row |
|---|---|---|---|---|---|---|---|
| 16 | 16 KiB | Direct Mapped | 8 B | 2 | 11 | 3 | 67 |
| 16 | 16 KiB | 2-way Set Associative | 8 B | 3 | 10 | 3 | 68 |
| 16 | 16 KiB | 4-way Set Associative | 8 B | 4 | 9 | 3 | 69 |
| 16 | 16 KiB | Fully Associative | 8 B | 13 | 0 | 3 | 78 |
| 32 | 64 KiB | Direct Mapped | 16 B | 16 | 12 | 4 | 145 |
| 32 | 64 KiB | Fully Associative | 16 B | 28 | 0 | 4 | 157 |
| 8 | 32 B | 4-way Set Associative | 4 B | 5 | 1 | 2 | 38 |

**How would these numbers change if the cache was write-back?**

Increase bits per row by one to account for the dirty bit.

## Cache Access Exercises

**Assume 16 B of memory and an 8B direct-mapped cache with 2-byte blocks. Classify each of the following memory accesses as hit (H), miss (M), or miss with replacement (R).**

For 16B memory, need 4 address bits. 2-byte blocks needs 1 offset bit. 4 cache rows needs 2 index bits.

a. 4= 0b0100 M (fills 0b10 index with M[4-5])
b. 5= 0b0101 H (finds block in 0b10 index, tag 0b0 matches)
c. 2= 0b0010 M (fills 0b01 index with M[2-3])
d. 6= 0b0110 M (fills 0b11 index with M[6-7])
e. 1= 0b0001 M (fills 0b00 index with M[0-1])
f. 10= 0b1010 R (tag 0b1 does not match 0b01 index, replaces with M[10-11])
g. 7= 0b0111 H (finds block in 0b11 index, tag 0b0 matches)
h. 2= 0b0010 R (tag 0b0 does not match 0b01 index, replaces with M[2-3])

**You know you have 1 MiB of memory (maxed out for processor address size) and a 16 KiB direct-mapped cache (data size only, not counting extra bits) with 1 KiB blocks.**

```
#define NUM_INTS 8192
int i, total = 0;
int *A = malloc(NUM_INTS * sizeof(int));    //returns address 0x100000
if(!A) return;
for (i = 0; i < NUM_INTS; i += 128) A[i] = i;        //Line 1
for (i = 0; i < NUM_INTS; i += 128) total += A[i];   //Line 2
```

**a) What is the T:I:O breakup for the cache (assuming byte addressing)?**

1 MiB memory = 20 address bits, 1 KiB blocks = 10 offset bits, 16 cache rows = 4 index bits. The TIO breakup is 6 : 4 : 10.

**b) Calculate the hit percentage for the cache for the line marked "Line 1".**

Each step of the loop jumps 128 ints (512 bytes). Each cache block holds 1 KiB (256 ints). This means that every other iteration will request a new block, so we have a 50% hit rate.

**c) Calculate the hit percentage for the cache for the line marked "Line 2".**

In total, the loop at Line 1 covers an array of size 8192*sizeof(int) = $2^{13} \times 2^2 = 2^{15}$ = 32 KiB. This means that our cache, which is only 16 KiB in size, had all of its rows overridden exactly once. Since Line 2 uses the same step size and same initial array index, we get the same hit rate as before (50%).

**d) How could you optimize the computation?**

We could have fewer cache misses if we break up the loops to cover half the array (16 KiB) at a time. While this means that there would be 4 for loops in our C code, the hit rates of each loop would be 50%, 100%, 50%, 100% for an average hit rate of 75%.

## Cache Parameters & Performance

How would making each of the changes affect cache performance and **why**?

| Parameter to Increase | Decreases | Increases |
|---|---|---|
| Cache Size | Miss rate (fewer capacity miss) | Hit time (more wires) |
| Block Size | Miss rate (more spatial locality) | Miss penalty (slower to load block) |
| Associativity | Miss rate (fewer conflict misses) | Hit time (more wires) |

**The miss rate changes are only rules of thumb—it's always possible to design an access pattern that will have the opposite effect!**