notes originally by Matt Johnson

More Pipelining Topics!

Hazards: Our pipelined execution doesn't always go smoothly. There are three specific types of conflicts that can arise: what are they?

Superscalar Hardware: Some pipeline problems happen when we don't have enough hardware (e.g. "If only I had two dryers…"). Superscalar means adding that redundant hardware for some instruction-level parallelism! Would that help latency or throughput? Who decides what hardware is used when?

Out-of-Order Execution: After instruction fetch, dispatch to a queue where the instruction waits until input operands are available. Queue is not always FIFO! Results are queued too, and write-back order happens in the "original" instruction order. Pretty complex! Low-end processors still do not use this paradigm due to the silicon area that is required for its implementation… In this class, OoOE would essentially refer to hardware doing the work of filling branch/load delay slots. The lecture slides show an instruction pausing in the middle of its execution.

Caches!

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

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

Break up an address:

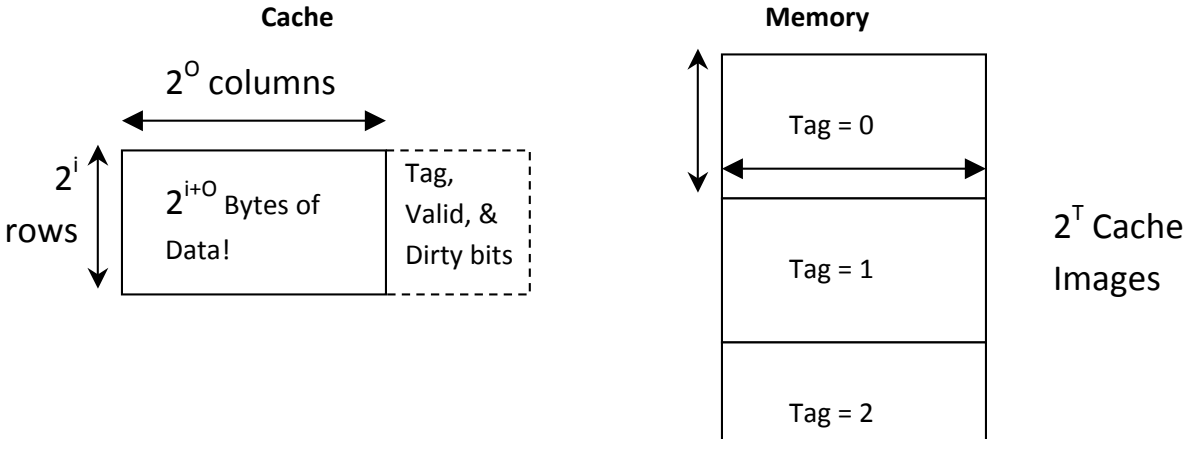
Tag	**I**ndex	**O**ffset

Offset: "column index" (O bits)

Index: "row index" (i bits)

Tag: "cache number" that the block/row* came from. (T bits) [*difference?]

Segmenting the address into TIO implies a geometrical structure (and size) on our cache. Draw memory with that same geometry!



notes originally by Matt Johnson

Cache Vocab:

Cache hit – found the right thing in the cache! Booyah!

Cache miss – Nothing in the cache block we checked, so read from memory and write to cache!

Cache miss, block replacement – We found a block, but it had the wrong tag!

Cache Exercises!

C1: Fill this one in… Everything here is Direct-Mapped!

Address Bits	Cache Size	Block Size	Tag Bits	Index Bits	Offset Bits	Bits per Row
16	4KB	4B				
16	16KB	8B				
32	8KB	8B				
32	32KB	16B				
32	64KB		16	12	4	146
32	512KB				5	
64		64B		14		
64	2048KB					1069

C2: 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).

a. 4
b. 5
c. 2
d. 6
e. 1
f. 10
g. 7
h. 2

notes originally by Matt Johnson

C3: This composite question was inspired by exam questions but NOT identical since the exam questions use associative caches. Direct-mapped here!

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

```
#define NUM_INTS 8192
int *A = malloc(NUM_INTS * sizeof(int)); // returns address 0x100000
int i, total = 0;
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)?

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

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

d) How could you optimize the computation?

Now a completely different setup… Your cache now has 8-byte blocks and 128 rows, and memory has 22 bit addresses. The `ARRAY_SIZE` is 4 MiB and `A` starts at a block boundary.

```
for (i = 0; i < (ARRAY_SIZE/STRETCH); i += 1) {
     for (j = 0; j < STRETCH; j += 1) sum     += A[i*STRETCH + j];
     for (j = 0; j < STRETCH; j += 1) product *= A[i*STRETCH + j];
}
```

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

b) What is the cache size (data only, no tag and extra bits) in bytes?

c) What is the largest `STRETCH` that minimizes cache misses?

d) Given the `STRETCH` size from (c), what is the # of cache misses?

e) Given the `STRETCH` size from (c), if `A` **does not** start at a block boundary, *roughly* what is the # of *cache misses* for this case to the number you calculated in question (d) above? (e.g., 8x, 1/16th)