CSCI 210: Computer Architecture
Cache Simulator Final Project

Stephen Checkoway
Oberlin College

Cache Simulator

- Take in a trace of load/stores from a real program
- Simulate running the program on a given cache configuration
- Calculate how well that configuration would perform for that trace

Cache Parameters

- Always: Write-allocate, write-back, LRU replacement

- Change:
  - Cache size
  - Block size
  - Associativity
  - Miss penalty

Store-hit policy: write-back

- Only update the block in cache
  - Keep track of whether each block is “dirty” (i.e., it has a different value than in memory)
- When a dirty block is replaced
  - Write it back to memory
  - Can use a write buffer to allow replacing block to be read first
- Faster than write-through, but more complex

V	D	Tag	Data
1	0	000042	FE FF 3C …
0			
1	1	001234	65 82 5C …
0			
0			
1	0	000F3C	00 00 00 …
0			
0			

Store-miss policy: write-allocate

- Read a block from memory into the cache (just like a load miss)
- Perform the write according to the store-hit policy (i.e., write in cache or write in both cache and memory)
- Good for when data is likely to be read shortly after being written (temporal locality)

Replacement Policy

- Least-recently used (LRU)
  - Choose the one unused for the longest time

Address Trace

Load/Store Address InstructionCount

```
# 0 7fffed80 1
# 0 10010000 10
# 0 10010060 3
# 1 10010030 4
# 0 10010004 6
# 0 10010064 3
# 1 10010034 4
```

L/S: 0 for load, 1 for store

Simulation Results

```
Simulation results:
    execution time                 52268708 cycles
    instructions                    5136716
    memory accesses                 1957764
    overall miss rate                  0.79
    load miss rate                     0.88
    CPI                               10.18
    average memory access time        24.07 cycles
    dirty evictions                  225876
    load_misses                     1525974
    store_misses                      30034
    load_hits                        205909
    store_hits                       195847
```

What do you need to do?

- Create data structures that emulates a cache
- For each instruction, find where it would go in the cache, check if it's already there
- Calculate number of miss penalty cycles, load misses, store misses, instructions, etc.
- Suggestion: Keep track of counts of events (e.g., load/store hits/misses, instruction count, dirty evictions) and use those counts to compute statistics like execution time, miss rates, etc.

Questions on Cache Final Project?

We have an 8 byte block size, direct-mapped, 16 kB cache. For a given byte address, to find the offset

A. Mod by 8

B. Mod by 16

C. Divide by 8

D. Divide by 16

E. None of the above

If we are simulating a cache and not actually accessing data, do we need to use the offset?

A. Yes

B. No

We have an 8 byte block size, direct-mapped, 16 kB cache. How many rows will this cache have?

A. 16

B. 2*1024

C. 8*1024

D. 16*1024

*Recall a KB is 1024 bytes

We have an 8 byte block size, direct-mapped, 16 KB cache. How can we find the index from an address?

A. Divide by 8

B. Divide by (16*1024)

C. Mod by 16*1024

D. Divide by 8, then mod by (2*1024)

E. None of the above

We have an 8 byte block size, **2-way set associative**, 16 kB cache. How can we find the index from an address?

A. Divide by 8, then mod by 1024

B. Divide by 8, then mod by (2*1024)

C. Divide by 8, then mod by (2*2*1024)

D. Divide by 16, then mod by (2*1024)

E. None of the above

We have an 8 byte block size, direct mapped, 16 KB cache. How can we find the tag of an address?

A. Divide by 8

B. Divide by 8 * 1024

C. Divide by 16*1024

D. None of the above