CS 61C: Great Ideas in Computer Architecture (Machine Structures) Caches Part I

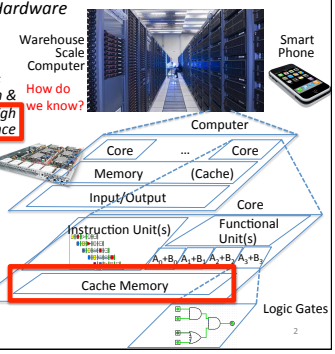
Instructors:
Krste Asanovic & Vladimir Stojanovic
http://inst.eecs.berkeley.edu/~cs61c/

New-School Machine Structures (It's a bit more complicated!)

Software
- Parallel Requests
  Assigned to computer
  e.g., Search "Katz"
- Parallel Threads
  Assigned to core
  e.g., Lookup, Ads
- Parallel Instructions
  >1 instruction @ one time
  e.g., 5 pipelined instructions
- Parallel Data
  >1 data item @ one time
  e.g., Add of 4 pairs of words
- Hardware descriptions
  All gates @ one time
- Programming Languages

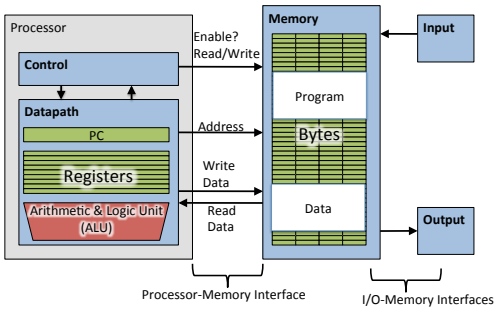
Harness Parallelism & Achieve High Performance

Hardware



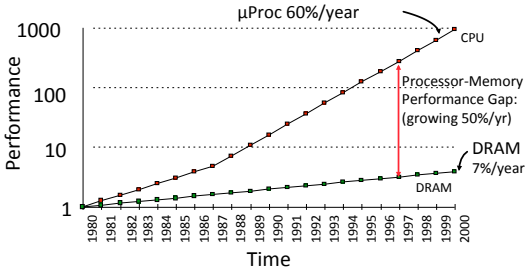
2

Components of a Computer



3

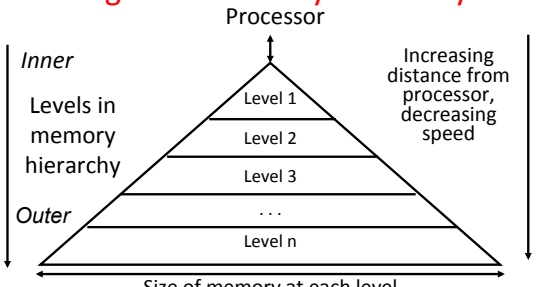
Processor-DRAM Gap (latency)



1980 microprocessor executes ~one instruction in same time as DRAM access
2015 microprocessor executes ~1000 instructions in same time as DRAM access

4

Big Idea: Memory Hierarchy



As we move to outer levels the latency goes up and price per bit goes down. Why?

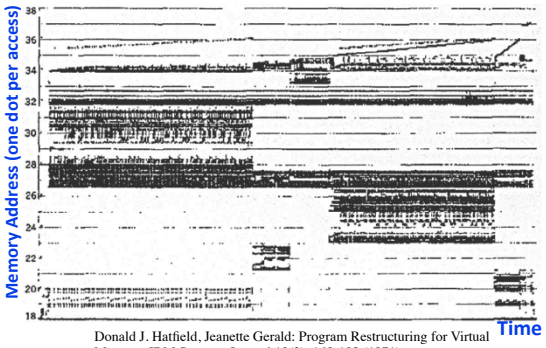
5

Library Analogy

- Writing a report based on books on reserve
  - E.g., works of J.D. Salinger
- Go to library to get reserved book and place on desk in library
- If need more, check them out and keep on desk
  - But don't return earlier books since might need them
- You hope this collection of ~10 books on desk enough to write report, despite 10 being only 0.00001% of books in UC Berkeley libraries

6

Real Memory Reference Patterns



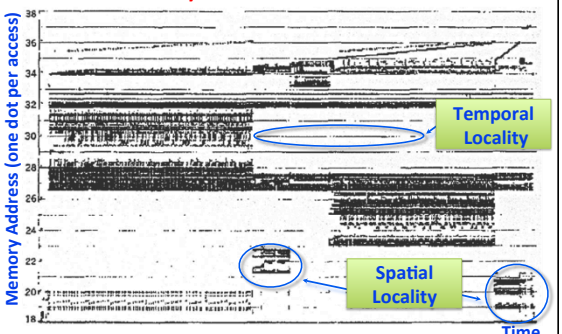
Donald J. Hatfield, Jeanette Gerald: Program Restructuring for Virtual Memory. IBM Systems Journal 10(3): 168-192 (1971)

Big Idea: Locality

- *Temporal Locality* (locality in time)
  - Go back to same book on desktop multiple times
  - If a memory location is referenced, then it will tend to be referenced again soon
- *Spatial Locality* (locality in space)
  - When go to book shelf, pick up multiple books on J.D. Salinger since library stores related books together
  - If a memory location is referenced, the locations with nearby addresses will tend to be referenced soon

8

Memory Reference Patterns



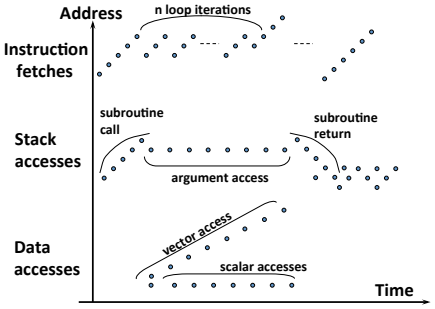
Donald J. Hatfield, Jeanette Gerald: Program Restructuring for Virtual Memory. IBM Systems Journal 10(3): 168-192 (1971)

Principle of Locality

- *Principle of Locality*: Programs access small portion of address space at any instant of time
- What program structures lead to temporal and spatial locality in instruction accesses?
- In data accesses?

10

Memory Reference Patterns



Cache Philosophy

- Programmer-invisible hardware mechanism to give illusion of speed of fastest memory with size of largest memory
  - Works fine even if programmer has no idea what a cache is
  - However, performance-oriented programmers today sometimes "reverse engineer" cache design to design data structures to match cache
  - We'll do that in Project 3

12

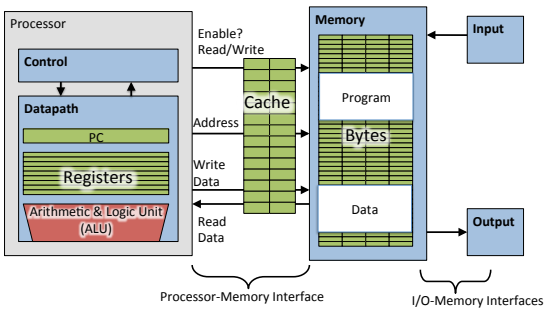
Memory Access without Cache

- Load word instruction: `lw $t0,0($t1)`
- $t1 contains 1022_{ten}, Memory[1022] = 99

1. Processor issues address 1022_{ten} to Memory
2. Memory reads word at address 1022_{ten} (99)
3. Memory sends 99 to Processor
4. Processor loads 99 into register $t0

13

Adding Cache to Computer



14

Memory Access with Cache

- Load word instruction: `lw $t0,0($t1)`
- $t1 contains 1022_{ten}, Memory[1022] = 99
- With cache (similar to a hash)

1. Processor issues address 1022_{ten} to Cache
2. Cache checks to see if has copy of data at address 1022_{ten}
   - 2a. If finds a match (Hit): cache reads 99, sends to processor
   - 2b. No match (Miss): cache sends address 1022 to Memory
     - I. Memory reads 99 at address 1022_{ten}
     - II. Memory sends 99 to Cache
     - III. Cache replaces word with new 99
     - IV. Cache sends 99 to processor
3. Processor loads 99 into register $t0

15

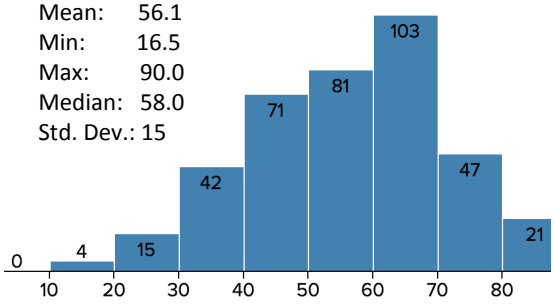
Administrivia

- Midterm 1 results out last week
- Project 2-1 due Sunday March 15th, 11:59PM
  - Use pinned Piazza threads!
  - We'll penalize those who ask, but don't search!
- Guerilla sections starting this weekend
  - Optional sections, focus on lecture/exam material, not projects
  - Vote for time on Piazza poll

16

Midterm Score Distribution

Mean: 56.1
Min: 16.5
Max: 90.0
Median: 58.0
Std. Dev.: 15



17

In the News: RowHammer Exploit

Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors

Yoongu Kim[1] Ross Daly* Jeremie Kim[1] Chris Fallin* Ji Hye Lee[1]
Donghyuk Lee[1] Chris Wilkerson[2] Konrad Lai Onur Mutlu[1]
[1]Carnegie Mellon University [2]Intel Labs

- CMU + Intel researchers found commercial DRAM chips susceptible to neighboring bits flipping if one row of memory accessed frequently
- Google Engineers figured out how to use this to gain root access on a machine! Almost all laptops susceptible, but server ECC memory helps reduce impact.

18

Cache "Tags"

- Need way to tell if have copy of location in memory so that can decide on hit or miss
- On cache miss, put memory address of block in "tag address" of cache block
  - 1022 placed in tag next to data from memory (99)

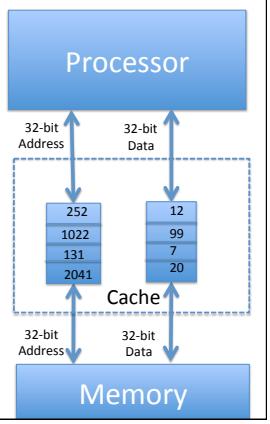
Tag	Data
252	12
1022	99
131	7
2041	20

From earlier instructions

19

Anatomy of a 16 Byte Cache, 4 Byte Block

- Operations:
  1. Cache Hit
  2. Cache Miss
  3. Refill cache from memory
- Cache needs Address Tags to decide if Processor Address is a Cache Hit or Cache Miss
  - Compares all 4 tags



Cache Replacement

- Suppose processor now requests location 511, which contains 11?
- Doesn't match any cache block, so must "evict" one resident block to make room
  - Which block to evict?
- Replace "victim" with new memory block at address 511

Tag	Data
252	12
1022	99
511	11
2041	20

21

Block Must be Aligned in Memory

- Word blocks are aligned, so binary address of all words in cache always ends in 00_{two}
- How to take advantage of this to save hardware and energy?
- Don't need to compare last 2 bits of 32-bit byte address (comparator can be narrower)

=> Don't need to store last 2 bits of 32-bit byte address in Cache Tag (Tag can be narrower)

22

Anatomy of a 32B Cache, 8B Block

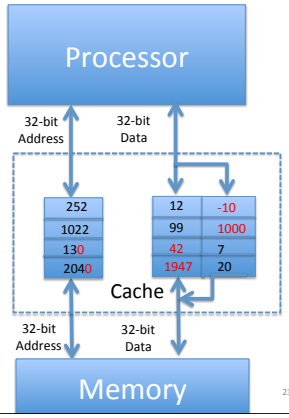
- Blocks must be aligned in pairs, otherwise could get same word twice in cache

⇒Tags only have even-numbered words

⇒ Last 3 bits of address always 000_{two}

⇒Tags, comparators can be narrower

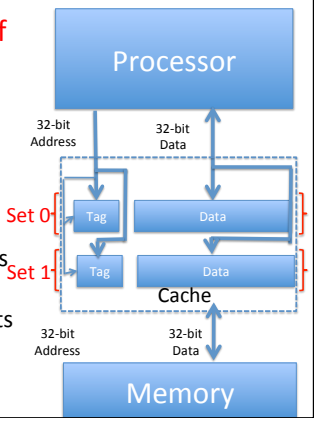
- Can get hit for either word in block



23

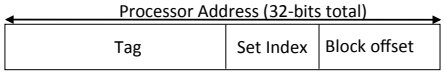
Hardware Cost of Cache

- Need to compare every tag to the Processor address
- Comparators are expensive
- Optimization: 2 sets => ½ comparators
- 1 Address bit selects which set



Processor Address Fields used by Cache Controller

- Block Offset: Byte address within block
- Set Index: Selects which set
- Tag: Remaining portion of processor address



- Size of Index = log2 (number of sets)
- Size of Tag = Address size – Size of Index – log2 (number of bytes/block)

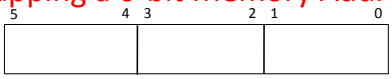
25

What is limit to number of sets?

- Can save more comparators if have more than 2 sets
- Limit: As Many Sets as Cache Blocks – only needs one comparator!
- Called "Direct-Mapped" Design

26

Mapping a 6-bit Memory Address



Mem Block Within $ Block *Tag*

Block Within $ *Index*

Byte Offset Within Block (e.g., Word)

- In example, block size is 4 bytes/1 word (it could be multi-word)
- Memory and cache blocks are the same size, unit of transfer between memory and cache
- # Memory blocks >> # Cache blocks
  - 16 Memory blocks/16 words/64 bytes/6 bits to address all bytes
  - 4 Cache blocks, 4 bytes (1 word) per block
  - 4 Memory blocks map to each cache block
- Byte within block: low order two bits, ignore! (nothing smaller than a block)
- Memory block to cache block, aka *index*: middle two bits
- Which memory block is in a given cache block, aka *tag*: top two bits

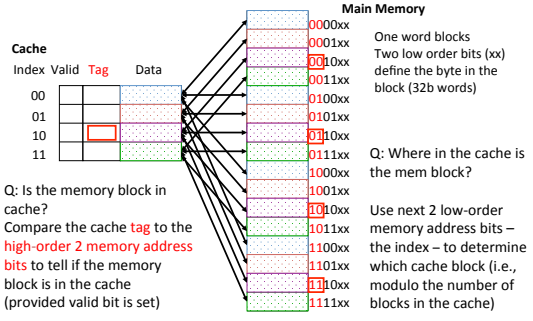
27

One More Detail: Valid Bit

- When start a new program, cache does not have valid information for this program
- Need an indicator whether this tag entry is valid for this program
- Add a "valid bit" to the cache tag entry
  - 0 => cache miss, even if by chance, address = tag
  - 1 => cache hit, if processor address = tag

28

Caching: A Simple First Example



One word blocks
Two low order bits (xx) define the byte in the block (32b words)

Q: Where in the cache is the mem block?

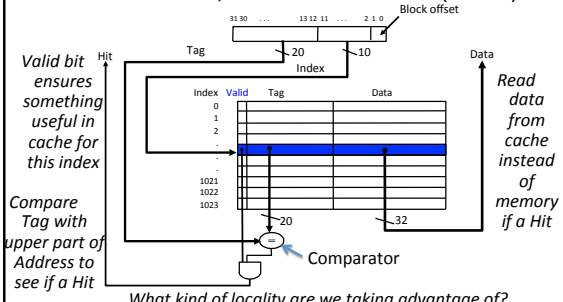
Use next 2 low-order memory address bits – the index – to determine which cache block (i.e., modulo the number of blocks in the cache)

Q: Is the memory block in cache?
Compare the cache tag to the high-order 2 memory address bits to tell if the memory block is in the cache (provided valid bit is set)

29

Direct-Mapped Cache Example

- One word blocks, cache size = 1K words (or 4KB)

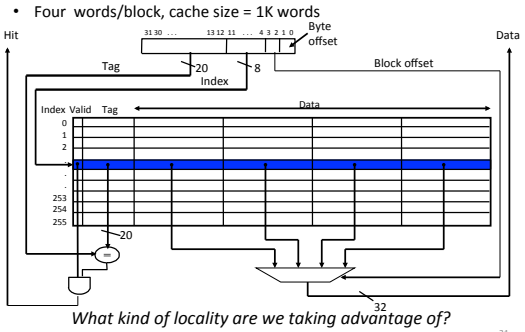


What kind of locality are we taking advantage of?

30

Multiword-Block Direct-Mapped Cache

- Four words/block, cache size = 1K words



What kind of locality are we taking advantage of?

31

Cache Names for Each Organization

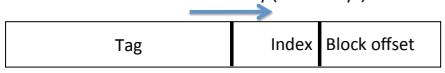
- "Fully Associative": Block can go anywhere
  - First design in lecture
  - Note: No Index field, but 1 comparator/block
- "Direct Mapped": Block goes one place
  - Note: Only 1 comparator
  - Number of sets = number blocks
- "N-way Set Associative": N places for a block
  - Number of sets = number of blocks / N
  - Fully Associative: N = number of blocks
  - Direct Mapped: N = 1

32

Range of Set-Associative Caches

- For a fixed-size cache, each increase by a factor of 2 in associativity doubles the number of blocks per set (i.e., the number of "ways") and halves the number of sets –
  - decreases the size of the index by 1 bit and increases the size of the tag by 1 bit

More Associativity (more ways)



Note: IBM persists in calling sets "ways" and ways "sets". They're wrong.

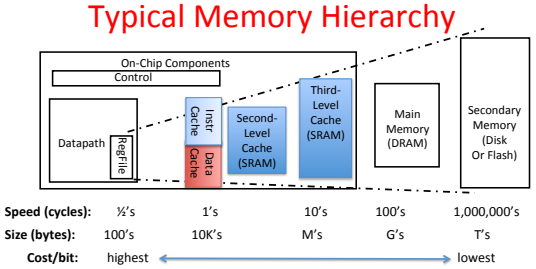
33

Clickers/Peer Instruction

- For a cache with constant total capacity, if we increase the number of ways by a factor of 2, which statement is false:
- A: The number of sets could be doubled
- B: The tag width could decrease
- C: The number of tags could stay the same
- D: The block size could be halved
- E: Tag width must increase

34

Typical Memory Hierarchy



- Principle of locality + memory hierarchy presents programmer with ≈ as much memory as is available in the *cheapest* technology at the ≈ speed offered by the *fastest* technology

35

Handling Stores with Write-Through

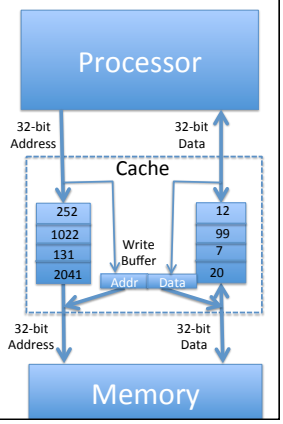
- Store instructions write to memory, changing values
- Need to make sure cache and memory have same values on writes: 2 policies

1) Write-Through Policy: write cache and write *through* the cache to memory
   - Every write eventually gets to memory
   - Too slow, so include Write Buffer to allow processor to continue once data in Buffer
   - Buffer updates memory in parallel to processor

36

Write-Through Cache

- Write both values in cache and in memory
- Write buffer stops CPU from stalling if memory cannot keep up
- Write buffer may have multiple entries to absorb bursts of writes
- What if store misses in cache?



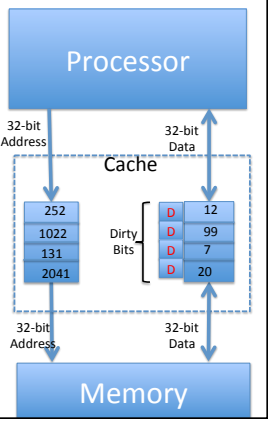
Handling Stores with Write-Back

2) Write-Back Policy: write only to cache and then write cache block *back* to memory when evict block from cache
- Writes collected in cache, only single write to memory per block
- Include bit to see if wrote to block or not, and then only write back if bit is set
  - Called "Dirty" bit (writing makes it "dirty")

38

Write-Back Cache

- Store/cache hit, write data in cache *only* & set dirty bit
  - Memory has stale value
- Store/cache miss, read data from memory, then update and set dirty bit
  - "Write-allocate" policy
- Load/cache hit, use value from cache
- On any miss, write back evicted block, only if dirty. Update cache with new block and clear dirty bit.



Write-Through vs. Write-Back

- Write-Through:
  - Simpler control logic
  - More predictable timing simplifies processor control logic
  - Easier to make reliable, since memory always has copy of data (big idea: Redundancy!)
- Write-Back
  - More complex control logic
  - More variable timing (0,1,2 memory accesses per cache access)
  - Usually reduces write traffic
  - Harder to make reliable, sometimes cache has only copy of data

40

And In Conclusion, …

- Principle of Locality for Libraries /Computer Memory
- Hierarchy of Memories (speed/size/cost per bit) to Exploit Locality
- Cache – copy of data lower level in memory hierarchy
- Direct Mapped to find block in cache using Tag field and Valid bit for Hit
- Cache design choice:
  - Write-Through vs. Write-Back

41