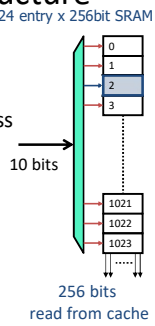
Summary so far

- Things we've covered:
  - The Need for Speed
  - Locality to the Rescue!
  - Calculating average memory access time
  - $ Misses: Cold, Conflict, Capacity
  - $ Characteristics: Associativity, Block Size, Capacity
- Things we skipped (and are about to cover):
  - Cache Overhead
  - Replacement Policies
  - Writes

61

Basic Memory Array Structure

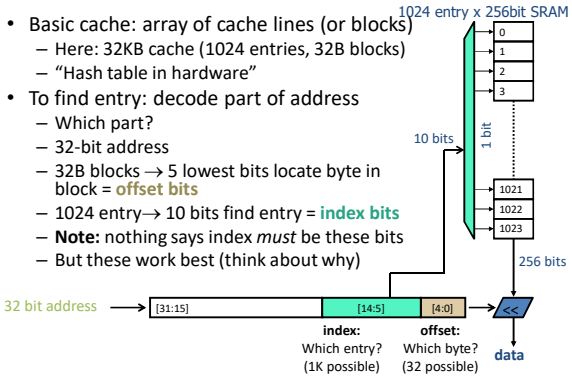
- Number of entries
  - n bits for lookup → 2^n entries
  - Example: 1024 entries, 10 bit address
  - Decoder changes n-bit address to 2^n bit "one-hot" signal
- Size of entries
  - Width of data accessed
  - Here: 256 bits (32 bytes)



62

Caches: Finding Data via Indexing

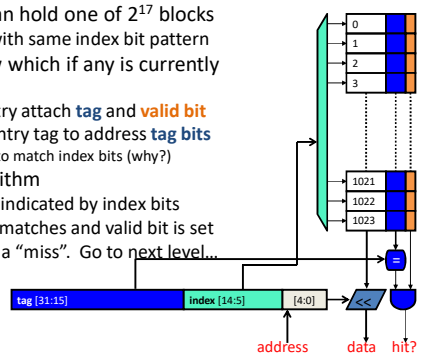
- Basic cache: array of cache lines (or blocks)
  - Here: 32KB cache (1024 entries, 32B blocks)
  - "Hash table in hardware"
- To find entry: decode part of address
  - Which part?
  - 32-bit address
  - 32B blocks → 5 lowest bits locate byte in block = **offset bits**
  - 1024 entry→ 10 bits find entry = **index bits**
  - **Note:** nothing says index *must* be these bits
  - But these work best (think about why)



63

Knowing that You Found It: Tags

- Each entry can hold one of 2^{17} blocks
  - All blocks with same index bit pattern
- How to know which if any is currently there?
  - To each entry attach **tag** and **valid bit**
  - Compare entry tag to address **tag bits**
    - No need to match index bits (why?)
- Lookup algorithm
  - Read entry indicated by index bits
  - "Hit" if tag matches and valid bit is set
  - Otherwise, a "miss". Go to next level…



64

Calculating Tag Overhead

- "32KB cache" means cache holds 32KB of **data**
  - Called **capacity**
  - Tag storage is considered overhead
- Tag overhead of 32KB cache with 1024 x 32B entries
  - 32B blocks → ??-bit offset
  - 1024 entries → ??-bit index
  - 32-bit address → ??-bit tag
- What about 64-bit addresses?

65

Calculating Tag Overhead

- "32KB cache" means cache holds 32KB of **data**
  - Called **capacity**
  - Tag storage is considered overhead
- Tag overhead of 32KB cache with 1024 x 32B entries
  - 32B blocks → 5-bit offset
  - 1024 entries → 10-bit index
  - 32-bit address → 32-bits – (5-bit offset + 10-bit index) = 17-bit tag
    (17-bit tag + 1-bit valid) x 1024 entries = 18Kb tags = 2.2KB tags
    ~6% overhead
- What about 64-bit addresses?
  - Tag increases to 49 bits, ~20% overhead

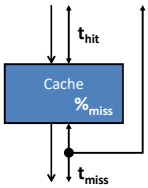
66

Handling a Cache Miss

- What if requested data isn't in the cache?
  - How does it get in there?
- **Cache controller**: finite state machine
  - Remembers miss address
  - Accesses next level of memory
  - Waits for response
  - Writes data/tag into proper locations
  - All of this happens on the **fill path**
  - Sometimes called **backside**

67

Cache Performance Equation



- **Access**: read or write to cache
- **Hit**: desired data found in cache
- **Miss**: desired data not found in cache
  - Must get from another component
  - No notion of "miss" in register file
- **Fill**: action of placing data into cache
- $\%_{miss}$ (miss-rate): #misses / #accesses
- t_{hit}: time to read data from (write data to) cache
- t_{miss}: time to read data into cache
- Performance metric: average access time

$$t_{avg} = t_{hit} + \%_{miss} \times t_{miss}$$

68

CPI Calculation with Cache Misses

- **Parameters**
  - Simple pipeline with base CPI of 1
  - Instruction mix: 30% loads/stores
  - I$: $\%_{miss}$ = 2%, t_{miss} = 10 cycles
  - D$: $\%_{miss}$ = 10%, t_{miss} = 10 cycles
- **What is new CPI?**
  - $CPI_{I\$}$ =
  - $CPI_{D\$}$ =
  - CPI_{new} =

69

CPI Calculation with Cache Misses

- **Parameters**
  - Simple pipeline with base CPI of 1
  - Instruction mix: 30% loads/stores
  - I$: $\%_{miss}$ = 2%, t_{miss} = 10 cycles
  - D$: $\%_{miss}$ = 10%, t_{miss} = 10 cycles
- **What is new CPI?**
  - $CPI_{I\$} = \%_{missI\$} \times t_{miss} = 0.02 \times 10 \text{ cycles} = 0.2 \text{ cycle}$
  - $CPI_{D\$} = \%_{load/store} \times \%_{missD\$} \times t_{missD\$} = 0.3 \times 0.1 \times 10 \text{ cycles} = 0.3 \text{ cycle}$
  - $CPI_{new} = CPI + CPI_{I\$} + CPI_{D\$} = 1+0.2+0.3 = 1.5$

70

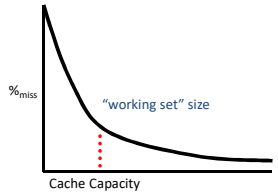
Measuring Cache Performance

- Ultimate metric is t_{avg}
  - Cache capacity and circuits roughly determines t_{hit}
  - Lower-level memory structures determine t_{miss}
  - Measure $\%_{miss}$
    - Hardware performance counters
    - Simulation

71

Capacity and Performance

- Simplest way to reduce $\%_{miss}$: increase capacity
  - \+ Miss rate decreases monotonically
    - **"Working set"**: insns/data program is actively using
    - Diminishing returns
  - However t_{hit} increases
    - Latency proportional to sqrt(capacity)
  - t_{avg} ?

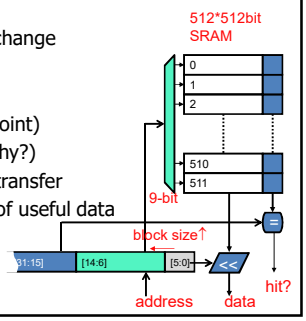


- Given capacity, manipulate $\%_{miss}$ by changing **organization**

75

Block Size

- Given capacity, manipulate $\%_{miss}$ by changing organization
- One option: increase **block size**
  - Exploit **spatial locality**
  - Notice index/offset bits change
  - Tag remain the same
- Ramifications
  - \+ Reduce $\%_{miss}$ (up to a point)
  - \+ Reduce tag overhead (why?)
  - – Potentially useless data transfer
  - – Premature replacement of useful data
  - – Fragmentation



76

Block Size and Tag Overhead

- Tag overhead of 32KB cache with 1024 32B entries
  - 32B lines → 5-bit offset
  - 1024 entries → 10-bit index
  - 32-bit address →
- Tag overhead of 32KB cache with 512 64B entries
  - 64B lines →
  - 512 entries →
  - 32-bit address →

77

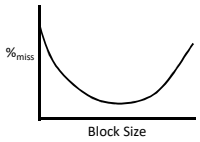
Block Size and Tag Overhead

- Tag overhead of 32KB cache with 1024 32B entries
  - 32B lines → 5-bit offset
  - 1024 entries → 10-bit index
  - 32-bit address → 32 – (5-bit offset + 10-bit index) = 17-bit tag
    (17-bit tag + 1-bit valid) x 1024 entries = 18Kb tags = 2.2KB tags
  - ~6% overhead
- Tag overhead of 32KB cache with 512 64B entries
  - 64B lines → 6-bit offset
  - 512 entries → 9-bit index
  - 32-bit address → 32 – (6-bit offset + 9-bit index) = 17-bit tag
    (17-bit tag + 1-bit valid) x 512 entries = 9Kb tags = 1.1KB tags
  - ~3% overhead

78

Effect of Block Size on Miss Rate

- Two effects on miss rate
  - \+ **Spatial prefetching (good)**
    - For blocks with adjacent addresses
    - Turns miss/miss into miss/hit pairs
  - – **Interference (bad)**
    - For blocks with non-adjacent addresses (but in adjacent entries)
    - Turns hits into misses by disallowing simultaneous residence
    - Consider entire cache as one big block
- Both effects always present
  - Spatial prefetching dominates initially
    - Depends on size of the cache
  - Good block size is 16–128B
    - Program dependent



80

Block Size and Miss Penalty

- Does increasing block size increase t_{miss}?
  - Don't larger blocks take longer to read, transfer, and fill?
  - They do, but…
- t_{miss} of an isolated miss is not affected
  - **Critical Word First / Early Restart (CRF/ER)**
  - Requested word fetched first, pipeline restarts immediately
  - Remaining words in block transferred/filled in the background
- t_{miss}'es of a cluster of misses will suffer
  - Reads/transfers/fills of two misses can't happen at the same time
  - Latencies can start to pile up
  - This is a bandwidth problem (more later)

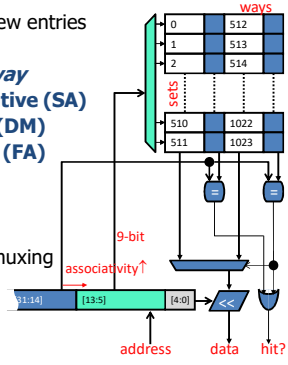
81

Set-Associativity

- Block can reside in one of few entries
- Entry groups called ***sets***
- Each entry in set called a ***way***
- This is **2-way set-associative (SA)**
- 1-way → **direct-mapped (DM)**
- 1-set → **fully-associative (FA)**

\+ Reduces conflicts
– Increases $latency_{hit}$:
  - additional tag match & muxing

- Note: valid bit not shown

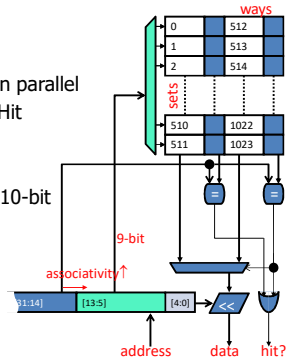


83

Set-Associativity

Lookup algorithm
- Use index bits to find set
- Read data/tags in all ways in parallel
- **Any** (match and valid bit), Hit

- Notice tag/index/offset bits
  - Only 9-bit index (versus 10-bit for direct mapped)
- Notice block numbering



84

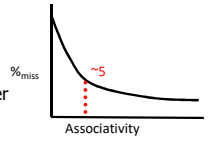
Replacement Policies

- Associative caches present a new design choice
  - On cache miss, which block in set to replace (kick out)?
- Some options
  - **Random**
  - **FIFO (first-in first-out)**
  - **LRU (least recently used)**
    - Fits with temporal locality, LRU = least likely to be used in future
  - **NMRU (not most recently used)**
    - An easier to implement approximation of LRU
    - Is LRU for 2-way set-associative caches
  - **Belady's**: replace block that will be used furthest in future
    - Unachievable optimum

87

Associativity and Performance

- Higher associative caches
  - + Have better (lower) $\%_{miss}$
    - Diminishing returns
  - – However t_{hit} increases
    - The more associative, the slower
  - What about t_{avg}?



- Block-size and number of sets should be powers of two
  - Makes indexing easier (just rip bits out of the address)
- 3-way set-associativity? No problem

88

Classifying Misses: 3C Model (Hill)

- Divide cache misses into three categories
  - **Compulsory (cold)**: never seen this address before
    - **Would miss even in infinite cache**
  - **Capacity**: miss caused because cache is too small
    - **Would miss even in fully associative cache**
    - Identify? Consecutive accesses to block separated by access to at least N other distinct blocks (N is number of entries in cache)
  - **Conflict**: miss caused because cache associativity is too low
    - Identify? **All other misses**
  - **(Coherence)**: miss due to external invalidations
    - Only in shared memory multiprocessors (later)
- Calculated by multiple simulations
  - Simulate infinite cache, fully-associative cache, normal cache
  - Subtract to find each count

89

Miss Rate: ABC

- Why do we care about 3C miss model?
  - So that we know what to do to eliminate misses
  - If you don't have conflict misses, increasing associativity won't help

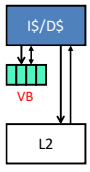
- **Associativity**
  - + Decreases conflict misses
  - – Increases $latency_{hit}$
- **Block size**
  - – Increases conflict/capacity misses (fewer entries)
  - + Decreases compulsory/capacity misses (spatial locality)
  - No significant effect on $latency_{hit}$
- **Capacity**
  - + Decreases capacity misses
  - – Increases $latency_{hit}$

90

Reducing Conflict Misses: Victim Buffer

- Conflict misses: not enough associativity
  - High-associativity is expensive, but also rarely needed
    - 3 blocks mapping to same 2-way set and accessed (XYZ)+

- **Victim buffer (VB)**: small fully-associative cache
  - Sits on I$/D$ miss path
  - Small so very fast (e.g., 8 entries)
  - Blocks kicked out of I$/D$ placed in VB
  - On miss, check VB: hit? Place block back in I$/D$
  - 8 extra ways, shared among all sets
    - + Only a few sets will need it at any given time
  - + Very effective in practice
  - Does VB reduce **$\%_{miss}$** or **$latency_{miss}$**?



91

Overlapping Misses: Lockup Free Cache

- **Lockup free**: allows other accesses while miss is pending
  - Consider: `load [r1]→r2; load [r3]→r4; add r2,r4 →r5`
  - **Handle misses in parallel**
    - "memory-level parallelism"
  - Makes sense for…
    - Processors that can go ahead despite D$ miss (out-of-order)
  - Implementation: **miss status holding register (MSHR)**
    - Remember: miss address, chosen entry, requesting instruction
    - When miss returns know where to put block, who to inform
  - Common scenario: "hit under miss"
    - Handle hits while miss is pending
    - Easy
  - Less common, but common enough: "miss under miss"
    - A little trickier, but common anyway
    - Requires multiple MSHRs: search to avoid frame conflicts

92