Memory Hierarchy and SRAM Cache Design

Mainak Chaudhuri
Indian Institute of Technology Kanpur

Sketch

- Abstract model of computer
- Locality principle
- Memory and storage hierarchy
- Basics of SRAM caches
  - Organization
  - Protocol for cache lookup
  - Cache hits and misses
- Multi-level cache hierarchy

2

Abstract model of computer

- Computer has an ISA
- The implementation of the ISA is an abstract five-state synchronous FSM
  - Each state change happens on posedge clock
  - State 0: fetch the instruction pointed to by program counter from memory; update program counter to point to the next instruction
  - State 1: decode the instruction to extract various fields and read source register operands
  - State 2: execute the instruction in ALU; compute address of load/store instructions; update program counter if control transfer instruction
  - State 3: access memory if load/store instruction
  - State 4: write result to destination register if the instruction produces a result

3

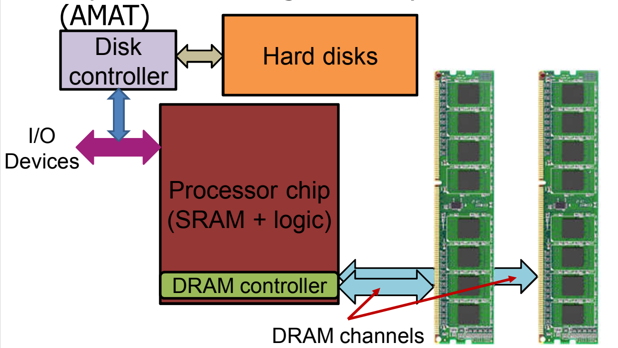
Abstract model of computer

- Fetching an instruction requires accessing memory with the program counter as addr.
- Decoding an instruction for MIPS is simple due to small number of formats and fixed position of the register specifiers
- Reading operands from register file requires exercising the read ports
- Executing an instruction and computing address of a load/store instruction requires an arithmetic logic unit (ALU)
- Load/store instructions access memory with the computed address
- Writing result to register file requires exercising the write ports

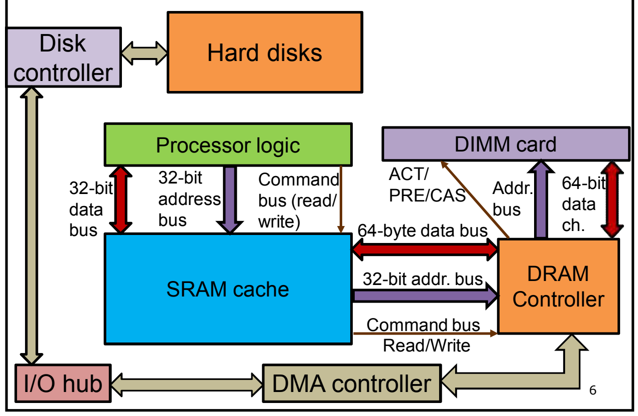
4

Abstract model of computer

- The SRAM cache on processor chip helps improve the average memory access time (AMAT)

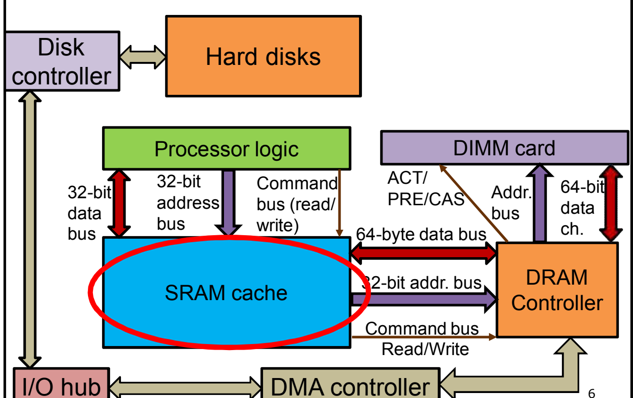


Abstract model of computer



6

Abstract model of computer



6

Locality principles

- Principles of locality exhibited by programs
  - Code and data accessed now are likely to be accessed again in near-future
    - Any interesting program would have loops and/or recursions
      - Justifies why code and data accesses may be repeated
      - Example: reuse of rows of A and columns of B when multiplying matrices A and B
    - Know as temporal locality
  - Code and data allocated close to the code and data being accessed now are likely to be accessed in near-future
    - Sequential code access
    - Sequential data access (e.g., walking over an array)
    - Known as spatial locality

7

Memory and storage hierarchy

- Locality principles imply an important corollary
  - Programs usually work on a small portions of code and data at a time
    - The code and data needed over a time window of length t could be a subset of the code and data needed over a bigger time window of length t'
    - Think about nested loops
- This corollary is exploited to build a hierarchy of memory and storage structures
  - Keep most recently used code and data close to the processor because this is needed now
  - Keep increasingly larger supersets of code and data gradually away from the processor

8

Memory and storage hierarchy

- Why not keep everything in a large on-chip SRAM?
  - Expensive and slow
- Memory and storage parts are usually arranged in a hierarchy
  - SRAM caches are closest to the processor logic, smallest in size, and fastest
    - Total on-chip cache is usually few tens of MBs
  - DRAM is outside processor chip, much larger in size, much slower than SRAM caches
    - Tens to hundreds of GBs
  - Hard disk holds everything, non-volatile, very large, very slow
    - Tens to hundreds of TBs

9

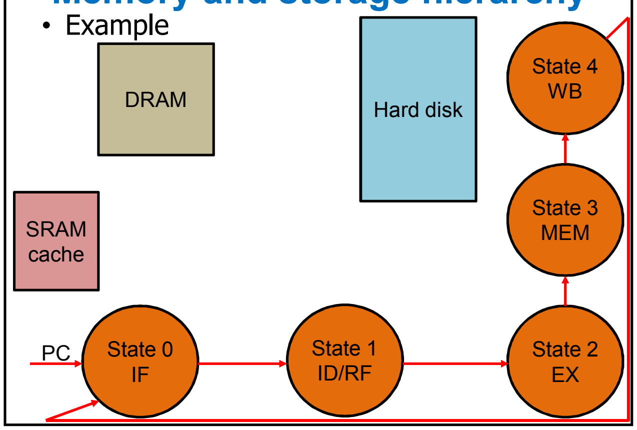
Memory and storage hierarchy

- Hierarchical organization allows very fast access to a small subset of code and data needed now from the SRAM cache
- Later this code and data can be exchanged to bring something else from DRAM
  - SRAM caches have finite capacity, so something must be replaced to bring something new if the cache is already full
- Also, code and data in DRAM can be swapped with something else from hard disk on demand
  - Less frequent than exchange between SRAM and DRAM

10

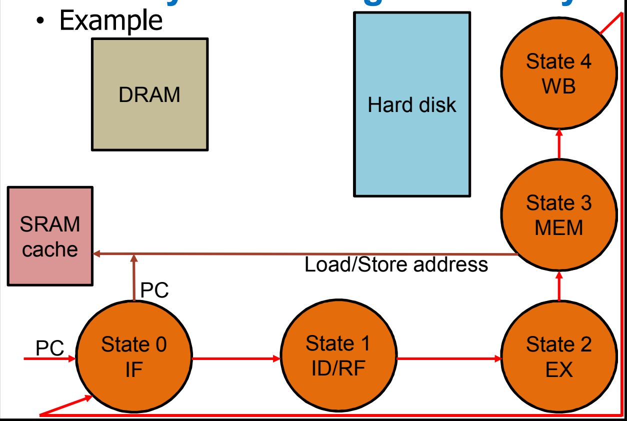
Memory and storage hierarchy

- Example



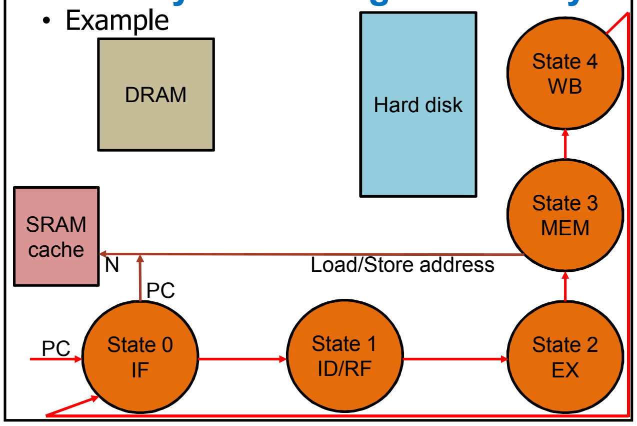
Memory and storage hierarchy

- Example



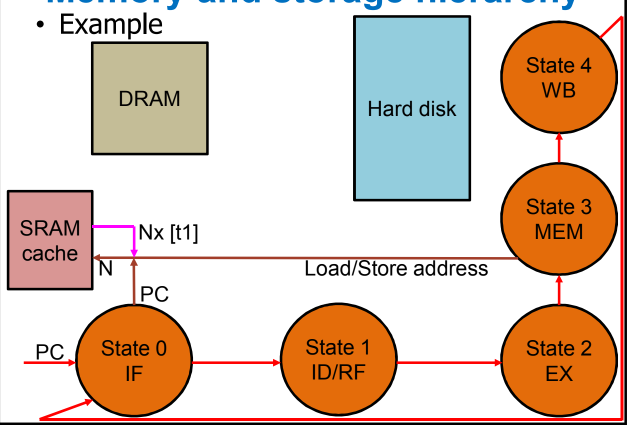
Memory and storage hierarchy

- Example



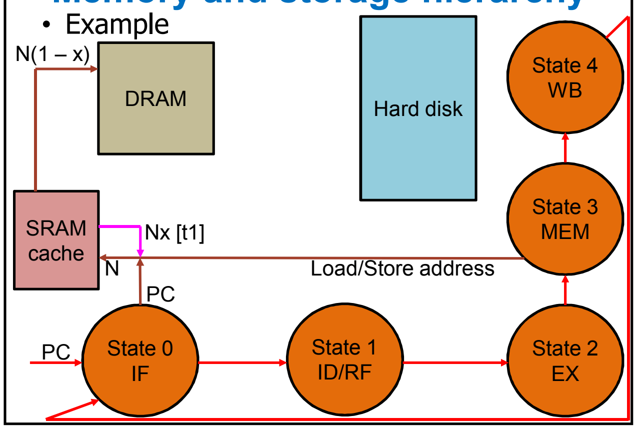
Memory and storage hierarchy

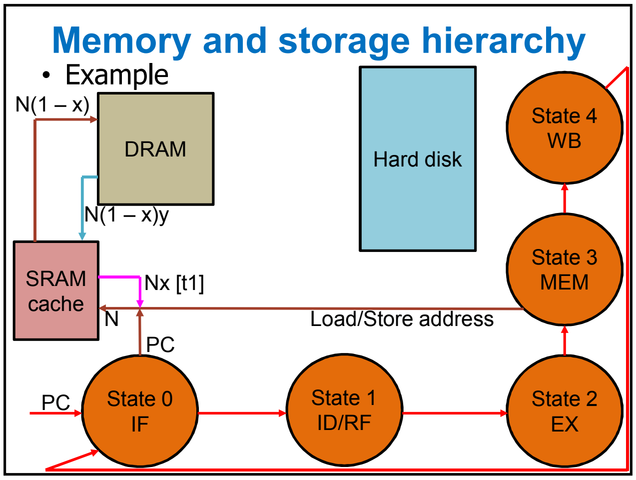
- Example

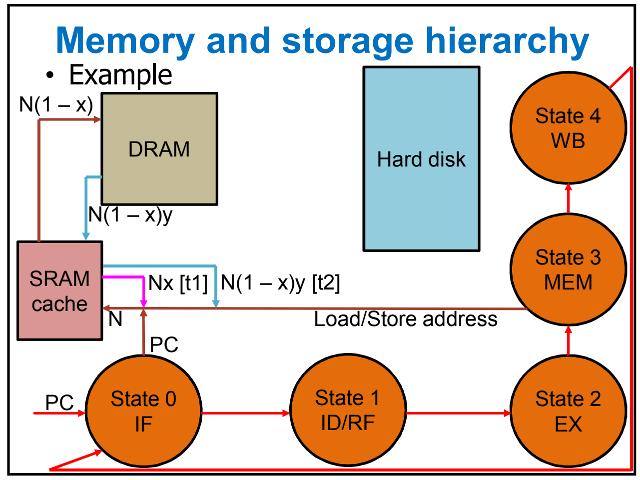


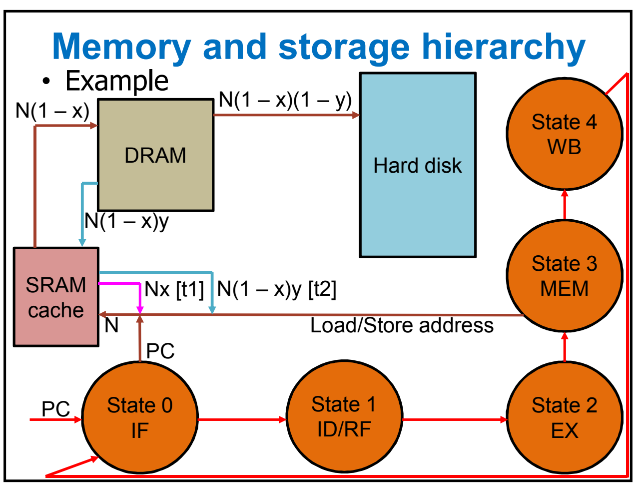
Memory and storage hierarchy

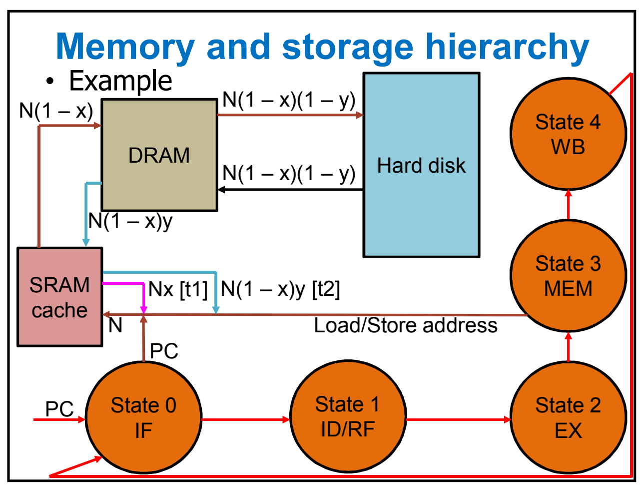
- Example




Memory and storage hierarchy
• Example
N(1 – x)
DRAM
N(1 – x)y
Hard disk
State 4
WB
State 3
MEM
SRAM
cache
Nx [t1]
N
Load/Store address
PC
PC
State 0
IF
State 1
ID/RF
State 2
EX

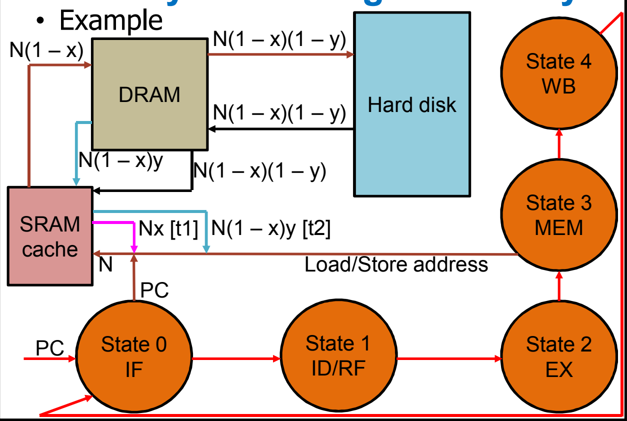

Memory and storage hierarchy
• Example
N(1 – x)
DRAM
N(1 – x)y
Hard disk
State 4
WB
State 3
MEM
SRAM
cache
Nx [t1]
N(1 – x)y [t2]
N
Load/Store address
PC
PC
State 0
IF
State 1
ID/RF
State 2
EX


Memory and storage hierarchy
• Example
N(1 – x)(1 – y)
N(1 – x)
DRAM
N(1 – x)y
Hard disk
State 4
WB
State 3
MEM
SRAM
cache
Nx [t1]
N(1 – x)y [t2]
N
Load/Store address
PC
PC
State 0
IF
State 1
ID/RF
State 2
EX


Memory and storage hierarchy
• Example
N(1 – x)(1 – y)
N(1 – x)
DRAM
N(1 – x)(1 – y)
N(1 – x)y
Hard disk
State 4
WB
State 3
MEM
SRAM
cache
Nx [t1]
N(1 – x)y [t2]
N
Load/Store address
PC
PC
State 0
IF
State 1
ID/RF
State 2
EX

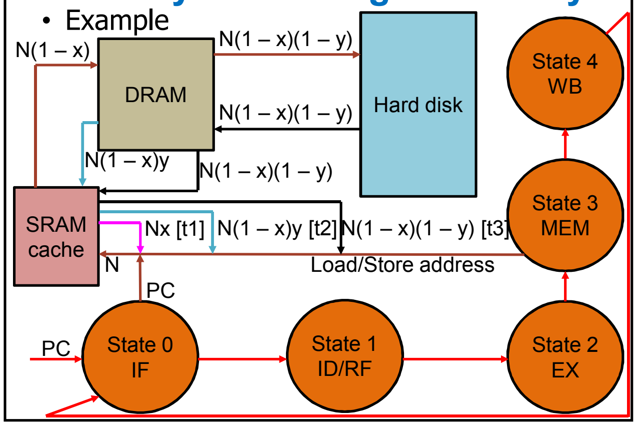
Memory and storage hierarchy

- Example



Memory and storage hierarchy

- Example



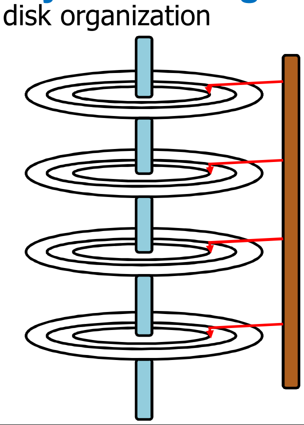
Memory and storage hierarchy

- Example
  - Suppose a program's load/store instructions and instruction fetcher generate N memory accesses
  - Nx accesses find the requested data in on-chip SRAM cache ($x < 1$)
  - $N(1 – x)y$ accesses find the requested data in DRAM ($y < 1$)
  - Remaining accesses fetch data from hard disk
  - An access to SRAM cache requires time t1 (hit)
  - An access to DRAM requires time t2 (cache miss)
  - An access to hard disk requires time t3
  - Average access time = $(Nxt1 + N(1 – x)yt2 + N(1 – x)(1 – y)t3)/N$
    - Since t1 << t2 << t3, as x and/or y increase(s), the average access time goes down

12

Memory and storage hierarchy

- Hard disk organization



Memory and storage hierarchy

- Hard disk organization
  - Metal platters are coated with magnetic recording material and arranged vertically on a spindle
    - One side or both sides of each platter can have recording capability
  - Each platter or disk has a head for reading and writing data
  - Each platter has concentric circles called tracks
  - All heads read from the same track of all platters at the same time
    - A particular track of all platters together forms a cylinder
  - Each track is divided into small contiguous data chunks called sectors (of size 512 to 4096 bytes)

Memory and storage hierarchy

- Hard disk organization
  - Anticipating good spatial locality in the running program, at a time a full sector is read out and filled in DRAM
    - The read sector necessarily contains the data that was demanded by the processor logic
  - Three components in accessing a target sector
    - Move the head assembly of all platters to the correct cylinder
      - Completely mechanical process and very slow
      - The time required is known as the seek latency
    - Rotate the platters to move the correct sector under the head
      - Also mechanical and slow; adds rotational latency
    - Read the sector and transfer to DRAM
      - Adds transfer latency determined by read & copy bandwidths

15

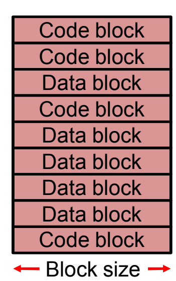
Basics of SRAM cache

- Start with a simple design
  - Array of code and data items
  - Any instruction or data element accessed will be fetched from DRAM and allocated a free slot in the cache, if it is not already in the cache
    - Hope is that due to temporal locality, this item will be accessed soon and at that time, it will be found in the cache resulting in much shorter access time
    - Drawback: doesn't exploit spatial locality
  - To exploit spatial locality, we could fetch a bigger block of code or data from DRAM containing the requested item
    - Mimics what the hard disk to DRAM interface does
    - Every SRAM cache fixes this fetch size and this is called the block size of the cache

16

Basics of SRAM cache

- Start with a simple design



17

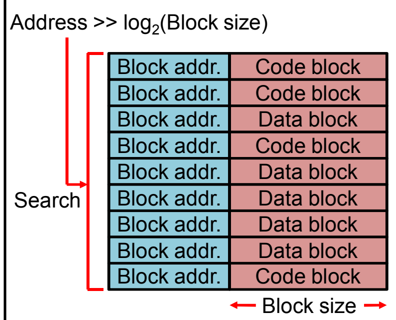
Basics of SRAM cache

- Cache is looked up in two events
  - State 0 of FSM generates an access for instruction using program counter as the address
  - State 3 of FSM generates an access for data using the address computed in state 2
  - Requested block is searched in the cache
    - Needs to store the address along with each block
      - Need to define block address (sequence number of a block)
      - If block size is 2^n bytes, block address is (Address >> n)
    - Block address is the search key
    - This searching time can be very large if done sequentially
    - A parallel search would require a large number of comparators (equal to number of blocks in the cache)
      - Would consume a lot of power and area

18

Basics of SRAM cache

- Start with a simple design



Address is either the PC or the load/store address

19

Basics of SRAM cache

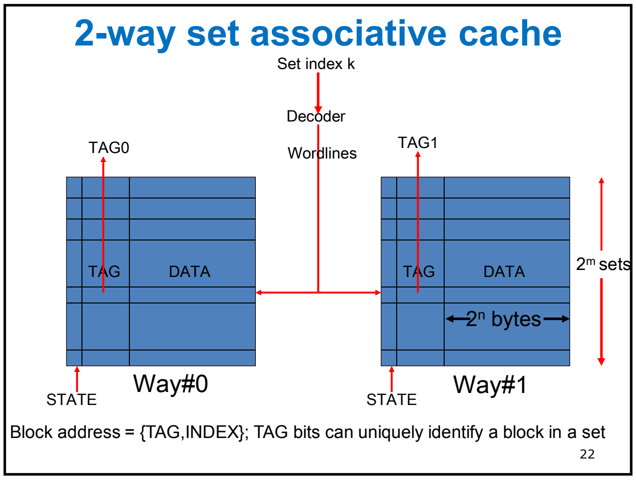
- If the looked up block is found in the cache, it is called a cache hit; otherwise it is a cache miss
  - Cache miss requests are forwarded to the DRAM controller for further handling
  - Eventually the DRAM controller will respond with the requested block and it will be allocated in the cache
    - What if the cache is full?
      - Needs to replace a block
    - Which block to replace?
      - Maybe the block that is not used recently (least-recently-used or LRU replacement algorithm)
      - Maybe a random block (random replacement algorithm)
      - LRU replacement requires keeping track of time of access
      - Random replacement requires a random number generator

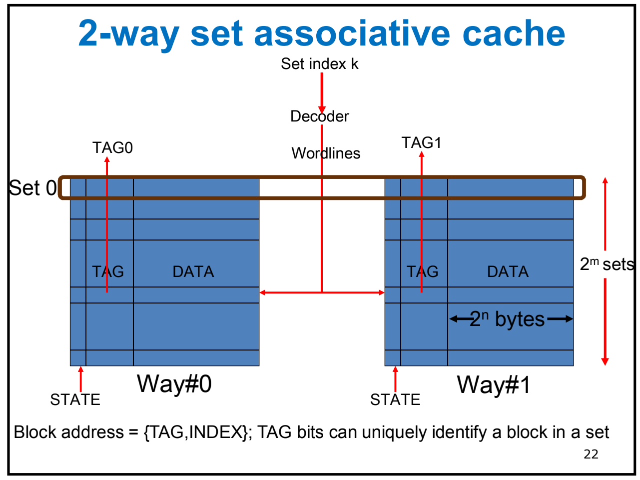
20

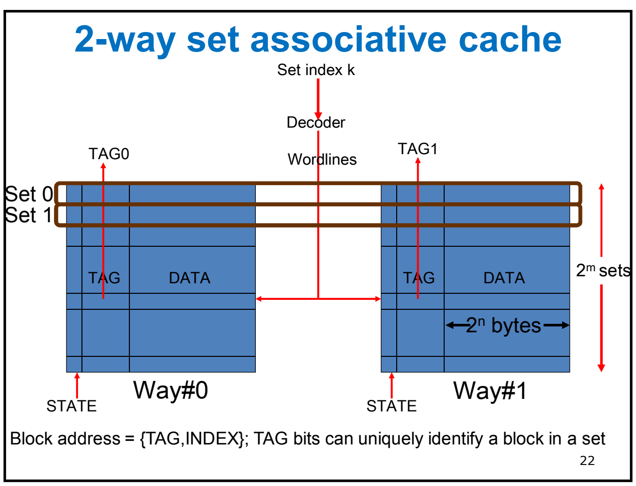
Basics of SRAM cache

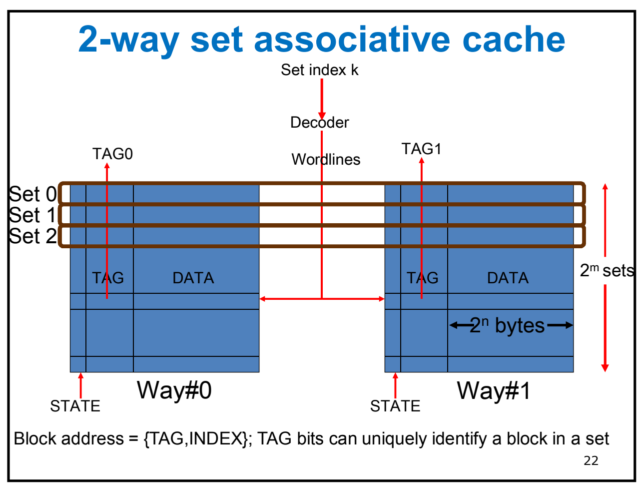
- Cache hits need to be much faster than cache misses to be useful
- To optimize the search time of a cache block, caches are typically organized as hash tables
  - Each hash element has a block, a block address or tag, and a few state bits (e.g., valid/invalid)
  - The blocks in a cache are logically divided into disjoint sets (these are hash buckets)
    - Each set can have a maximum number of valid blocks
    - This maximum number is known as the associativity of the cache
    - For example, a 16 KB cache with 64-byte blocks can have 32 sets each with associativity 8

21

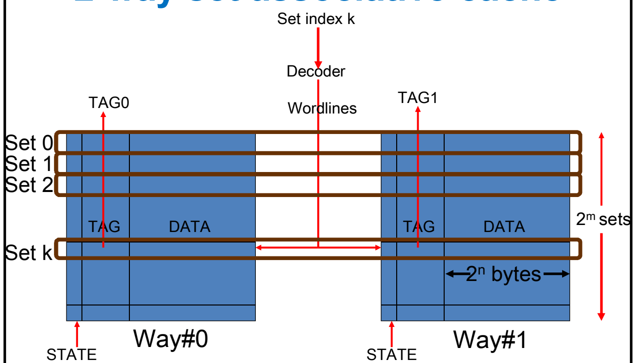

2-way set associative cache
Set index k
Decoder
Wordlines
TAG0
TAG1
TAG
DATA
TAG
DATA
2^m sets
2^n bytes
STATE
Way#0
STATE
Way#1
Block address = {TAG,INDEX}; TAG bits can uniquely identify a block in a set
22


2-way set associative cache
Set index k
Decoder
Wordlines
TAG0
TAG1
Set 0
TAG
DATA
TAG
DATA
2^m sets
2^n bytes
STATE
Way#0
STATE
Way#1
Block address = {TAG,INDEX}; TAG bits can uniquely identify a block in a set
22


2-way set associative cache
Set index k
Decoder
Wordlines
TAG0
TAG1
Set 0
Set 1
TAG
DATA
TAG
DATA
2^m sets
2^n bytes
STATE
Way#0
STATE
Way#1
Block address = {TAG,INDEX}; TAG bits can uniquely identify a block in a set
22


2-way set associative cache
Set index k
Decoder
Wordlines
TAG0
TAG1
Set 0
Set 1
Set 2
TAG
DATA
TAG
DATA
2^m sets
2^n bytes
STATE
Way#0
STATE
Way#1
Block address = {TAG,INDEX}; TAG bits can uniquely identify a block in a set
22

2-way set associative cache



Block address = {TAG,INDEX}; TAG bits can uniquely identify a block in a set

22

Basics of SRAM cache

- A cache with associativity A is often called an A-way set-associative cache
- To look up a cache with an address
  - The set index is first determined by passing the address through a hash function
  - Within the set, the block addresses are searched in parallel for the target address
    - Number of comparators is equal to the associativity of the cache (which is usually small)
  - Critical path of lookup: determine set index, decode set index, read all block addresses in a set, parallel comparison, select at most one data block based on comparison outcome (needs a multiplexor)

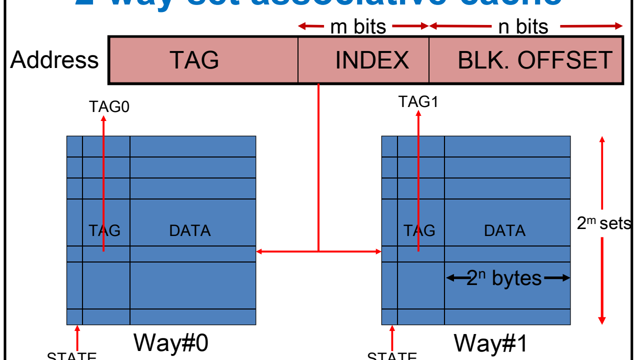
23

Basics of SRAM cache

- Determining set index
  - Many possible hash functions exist; we will discuss the simplest and the most popular one
  - Suppose there are N sets in the cache
  - To evenly distribute the block addresses among all sets, one possible mapping of block addresses to sets would map every N^{th} block address to the same set
    - Set 0 gets block addresses 0, N, 2N, 3N, …
    - Set 1 gets block addresses 1, N+1, 2N+1, 3N+1, …
  - Set index can be determined by (block address) mod N
    - N is always a power of two; therefore, (block address) mod N = (block address) & (N – 1) [avoids a division]

24

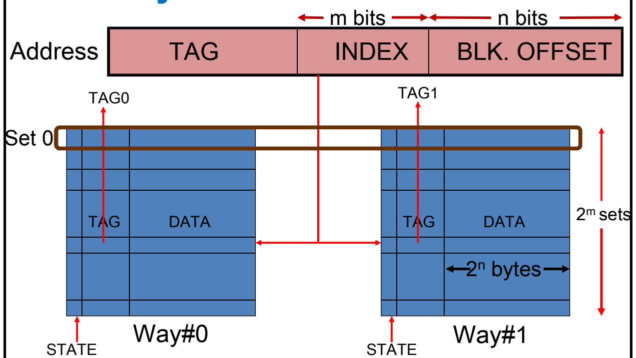
2-way set associative cache



Block address = {TAG,INDEX}; TAG bits can uniquely identify a block in a set
TAG0 and TAG1 are compared again TAG in parallel; at most one will match

25

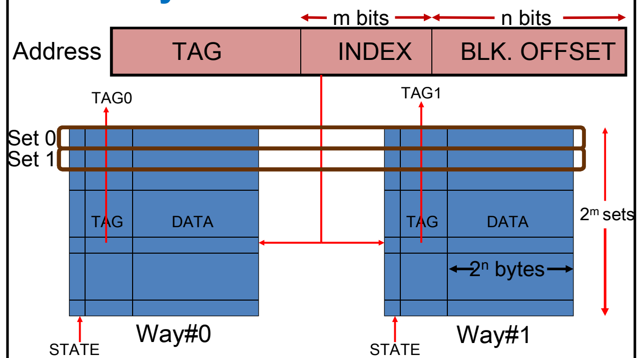
2-way set associative cache



Block address = {TAG,INDEX}; TAG bits can uniquely identify a block in a set
TAG0 and TAG1 are compared again TAG in parallel; at most one will match

25

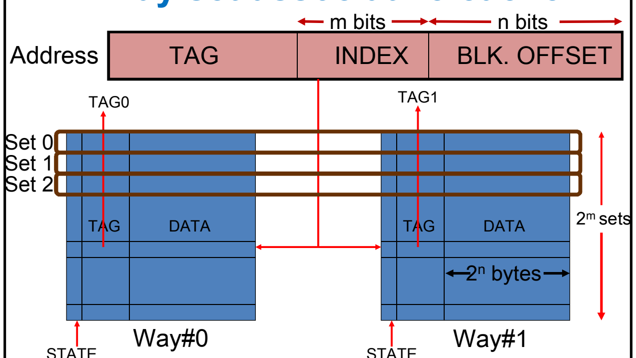
2-way set associative cache



Block address = {TAG,INDEX}; TAG bits can uniquely identify a block in a set
TAG0 and TAG1 are compared again TAG in parallel; at most one will match

25

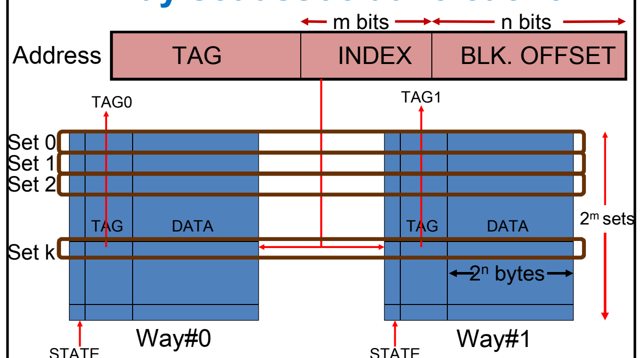
2-way set associative cache



Block address = {TAG,INDEX}; TAG bits can uniquely identify a block in a set
TAG0 and TAG1 are compared again TAG in parallel; at most one will match

25

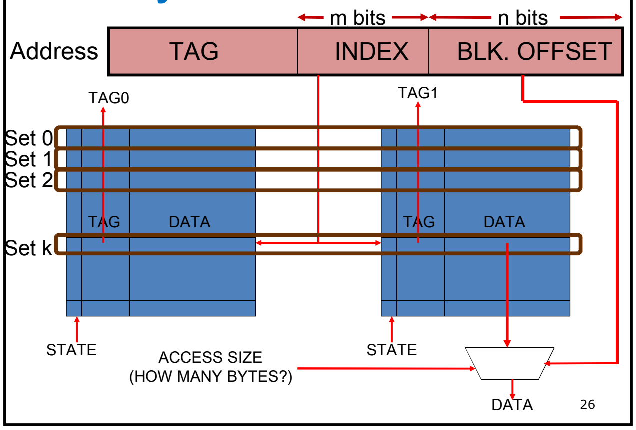
2-way set associative cache



Block address = {TAG,INDEX}; TAG bits can uniquely identify a block in a set
TAG0 and TAG1 are compared again TAG in parallel; at most one will match

25

2-way set associative cache



26

Basics of SRAM cache

- Observations
  - If associativity is one, number of sets is equal to the number of blocks in cache
    - A given block address has a unique location in cache
    - Known as direct-mapped cache (a given block address directly maps to a unique location in cache)
    - Needs a single comparator and no multiplexing needed for data selection on critical path
      - Faster hit time than set-associative designs
    - Simple design, but may suffer from large number of collisions between blocks (known as conflicts)
      - Block addresses 0, N, 2N, 3N, … all map to set 0 if there are N blocks in the cache
      - Can increase the number of cache misses compared to a set-associative design
    - This is the minimum possible associativity

27

Basics of SRAM cache

- Observations
  - If associativity is equal to the number of blocks in the cache, then the number of sets is one
    - Need to search all blocks in the cache during a lookup
      - Restricts the number of blocks in the cache; otherwise too many comparators and a large multiplexor would be needed consuming power and area
    - Known as fully associative cache (usually small in size)
    - Since number of sets is one, there are no index bits and the entire block address is the tag
    - Also, a given block can be placed anywhere in the cache
      - Significantly reduces conflicts
    - This is the maximum possible associativity for a given cache capacity
      - Cache capacity = no. of blocks x block size = no. of sets x no. of ways x block size

28

Basics of SRAM cache

- Total number of bits in cache
  - Data bits + Tag bits + valid bit = (Block size + Tag length + 1) x no. of sets x no. of ways
- Observation
  - For a fixed cache capacity, doubling of associativity halves number of sets
    - Decreases set index bits by one and increases tag length by one (total number of address bits is constant)
    - Number of comparators doubles, width of comparators increases by one bit, width of multiplexor increases by one bit
  - For a fixed cache capacity and associativity, doubling of block size also halves the no. of sets

29

Basics of SRAM cache

- Example
  - 32-bit address, block size 64 bytes, direct-mapped, number of blocks (or sets) 512. Calculate the total number of bits in cache
  - Tag length = 32 – 6 – 9 = 17 bits
  - Tag and valid bits = 512 x 18 bits
  - Total cache bits = 512 x (64x8 + 17 + 1) bits
- Example
  - 32 KB cache with 64-byte block size and two ways. Calculate the size of the set index decoder.
  - Number of sets = 32 x 2^{10}/(64 x 2) = 2^8
  - Set index decoder size = 8 bits to 256 bits

30

Basics of SRAM cache

- Example
  - 32 KB fully associative cache with 64-byte block size. Calculate the data selection multiplexor and comparator widths. Assume 32-bit address.
  - Number of ways = 32 x 2^{10}/64 = 2^9 = number of comparators
  - Width of multiplexor = 512 to 1
  - Tag length = 32 – 6 = 26 bits = comparator width

31

Multi-level cache hierarchy

- All commercial processors have two or three levels of SRAM cache on chip today
  - L1 is the closest to the processor and L2, L3, … are further away, get gradually bigger and slower
  - The levels are designed such that the latency of fetching a block from the last level SRAM cache is still much smaller than fetching from DRAM
    - Typical sizes: L1: < 100 KB, L2: < 1 MB, L3: 2-32 MB
    - Associativity typically increases with increasing level
    - Block size may increase or may remain constant
    - Typical round-trip latencies could be L1: <1 ns, L2: ~5 ns, L3: ~10 ns, DRAM: 50-100 ns
    - Missing in last-level cache can be highly detrimental for performance

32

Multi-level cache hierarchy

- Processor FSM and logic injects code/data requests to L1 instruction/data cache
- L1 cache hits are returned immediately to the processor; L1 cache misses are forwarded to the L2 cache
- L2 cache hits are returned to the L1 cache, which forwards the requested bytes to the processor and also fills the block in L1 cache
  - Future accesses can be satisfied from the L1 cache until the block is replaced from L1 cache
- L2 cache misses are forwarded to L3 cache or DRAM depending on the number of SRAM cache levels

33