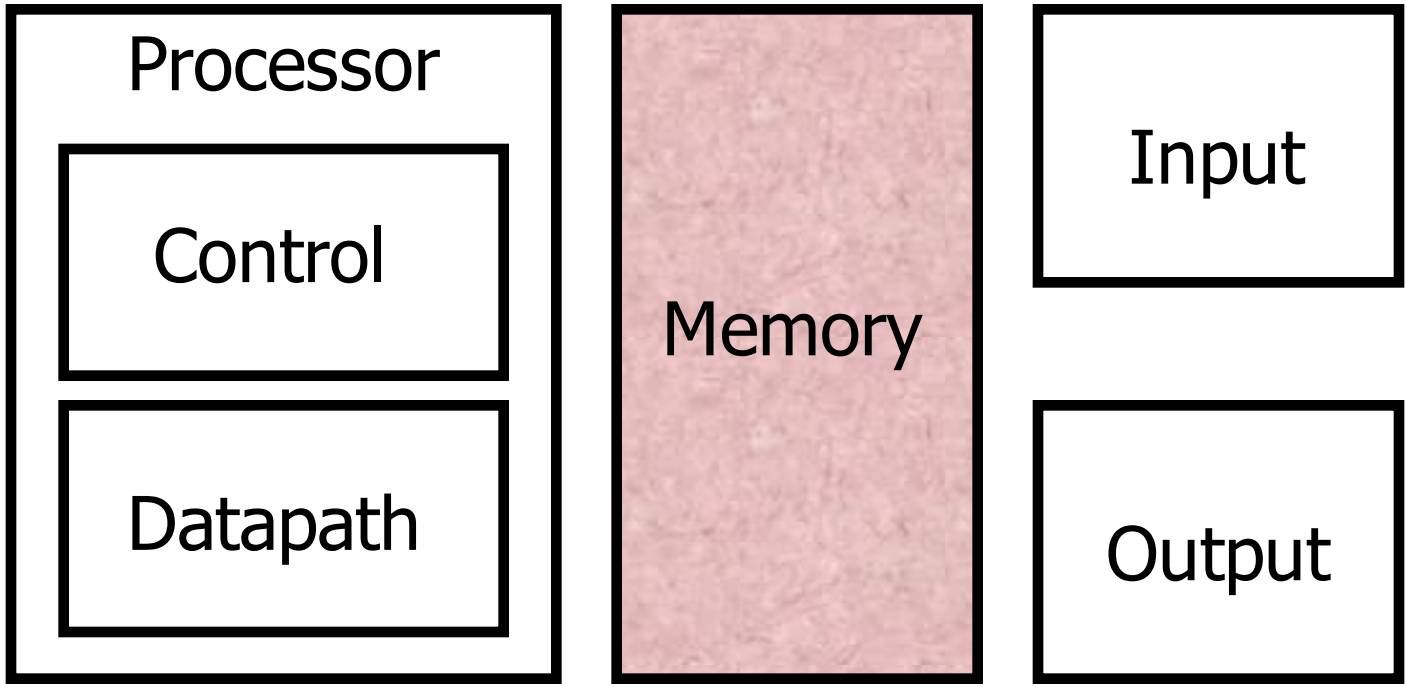
CC 311- Computer Architecture

CISC / RISC

Chapter 5
Large and Fast: Exploiting Memory Hierarchy

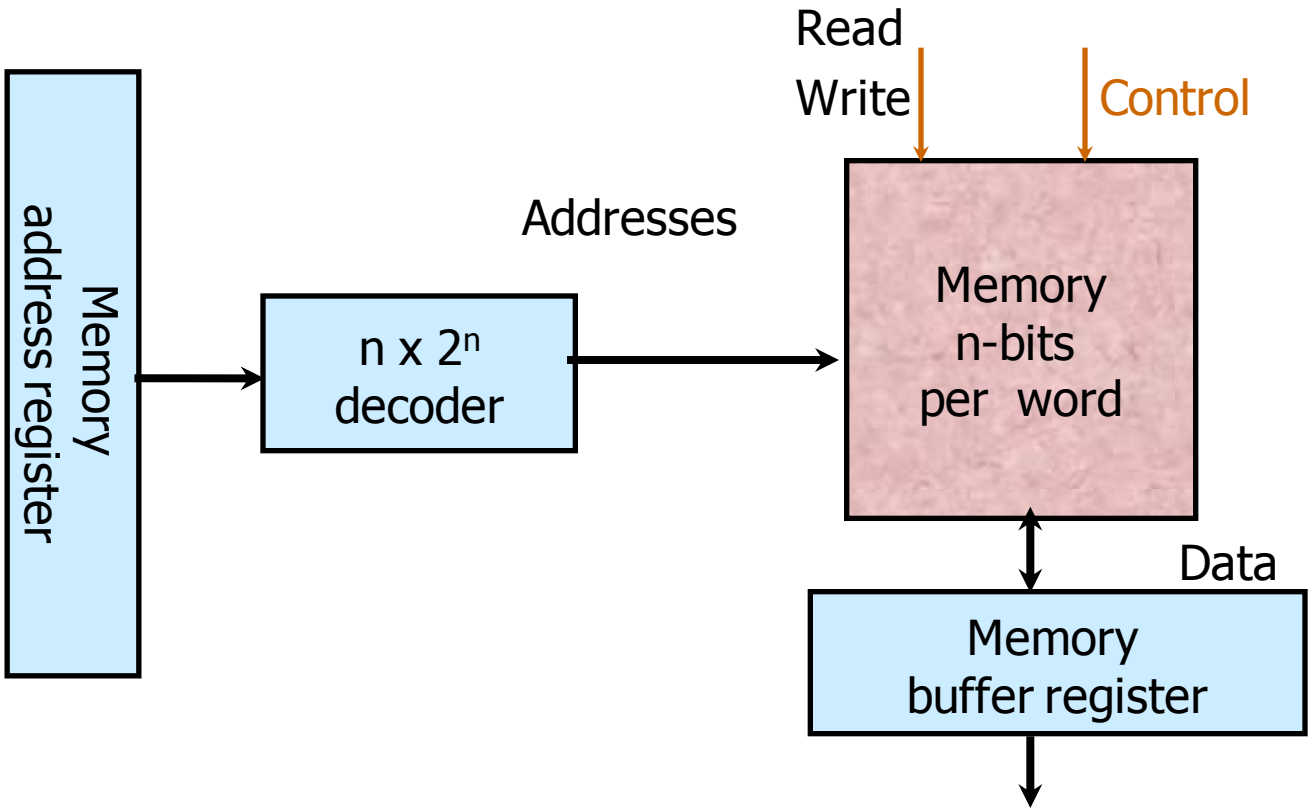
Introduction

- After CPU design, memory is next



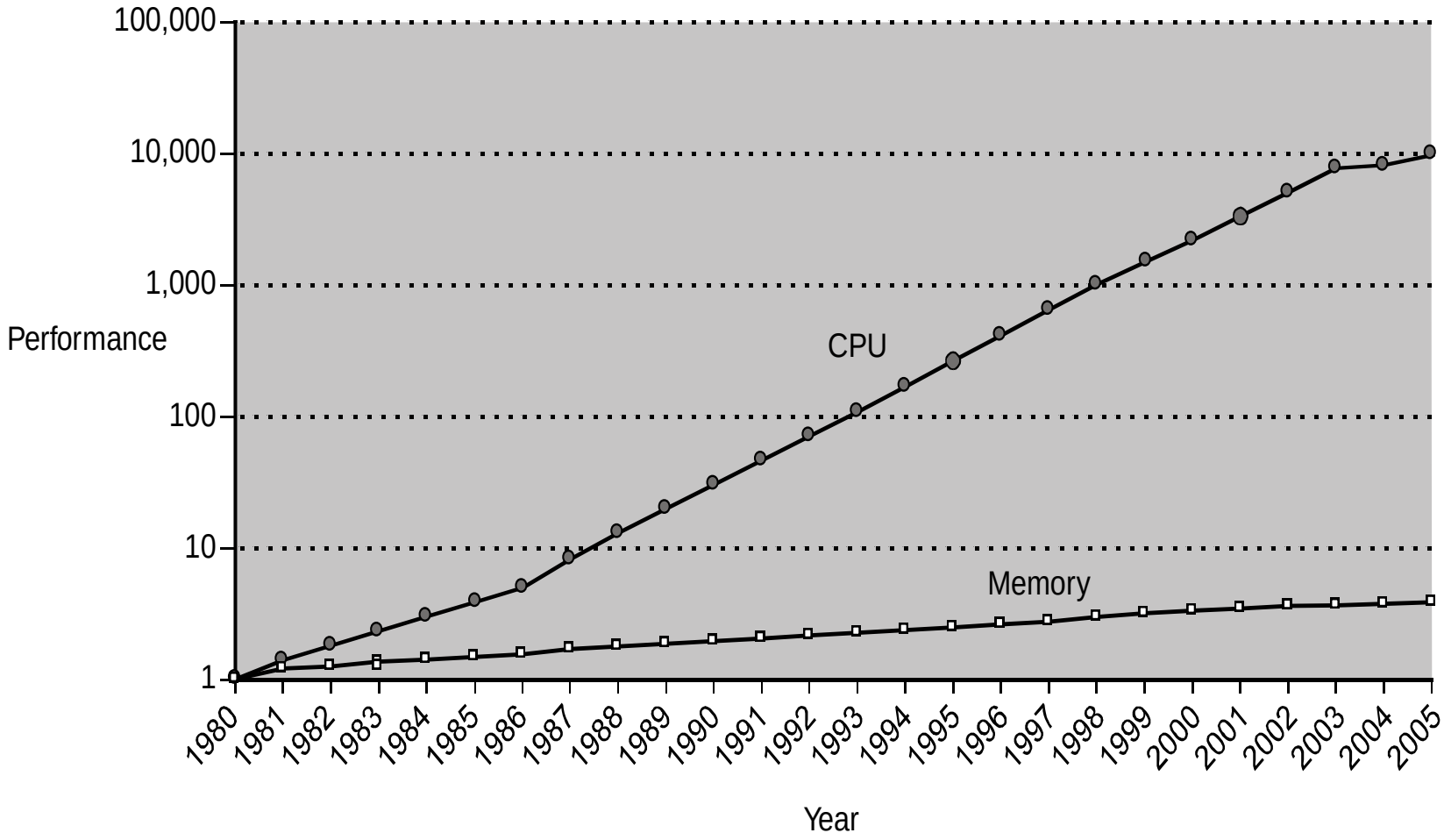
Memory Components

- Memory is Functionally a set of registers to hold programs & data



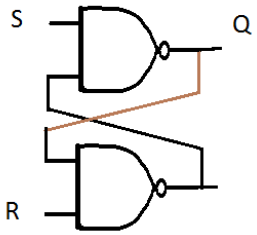
Some Issues

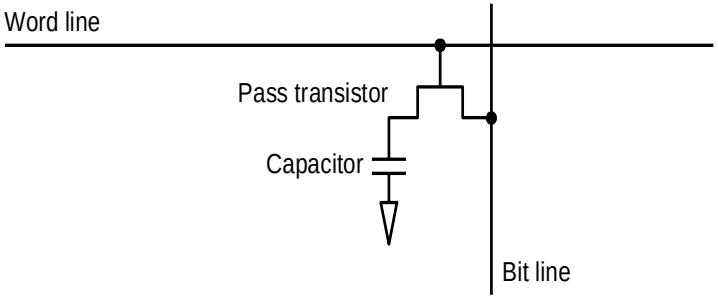
- Processor speeds continue to increase very fast



Memories: Review

- SRAM:
  - value is stored on a pair of inverting gates
  - very fast but takes up more space than DRAM (4 to 6 transistors)





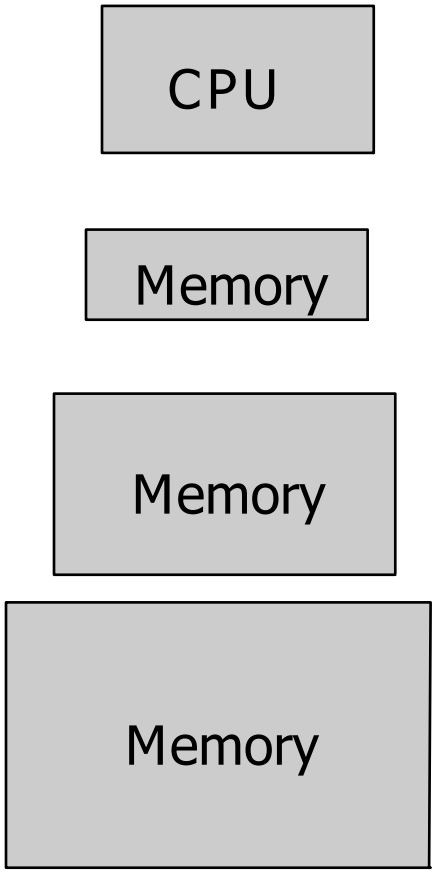
- DRAM:
  - value is stored as a charge on capacitor (must be refreshed)
  - Reading is destructive
  - very small but slower than SRAM (factor of 5 to 10)

Memory Hierarchy

- Users want large and fast memories!
- In 2004:
  - SRAM
    - Access times: .5 - 5ns
    - Cost: $4000 - $10,000 per GB.
  - DRAM
    - Access times: 50 - 70ns
    - Cost $100 - $200 per GB.
  - Disk
    - Access times: 5 - 20 million ns
    - Cost $.50 - $2 per GB.

Memory Hierarchy

- By implementing memory hierarchy, user has the illusion of large & fast memory



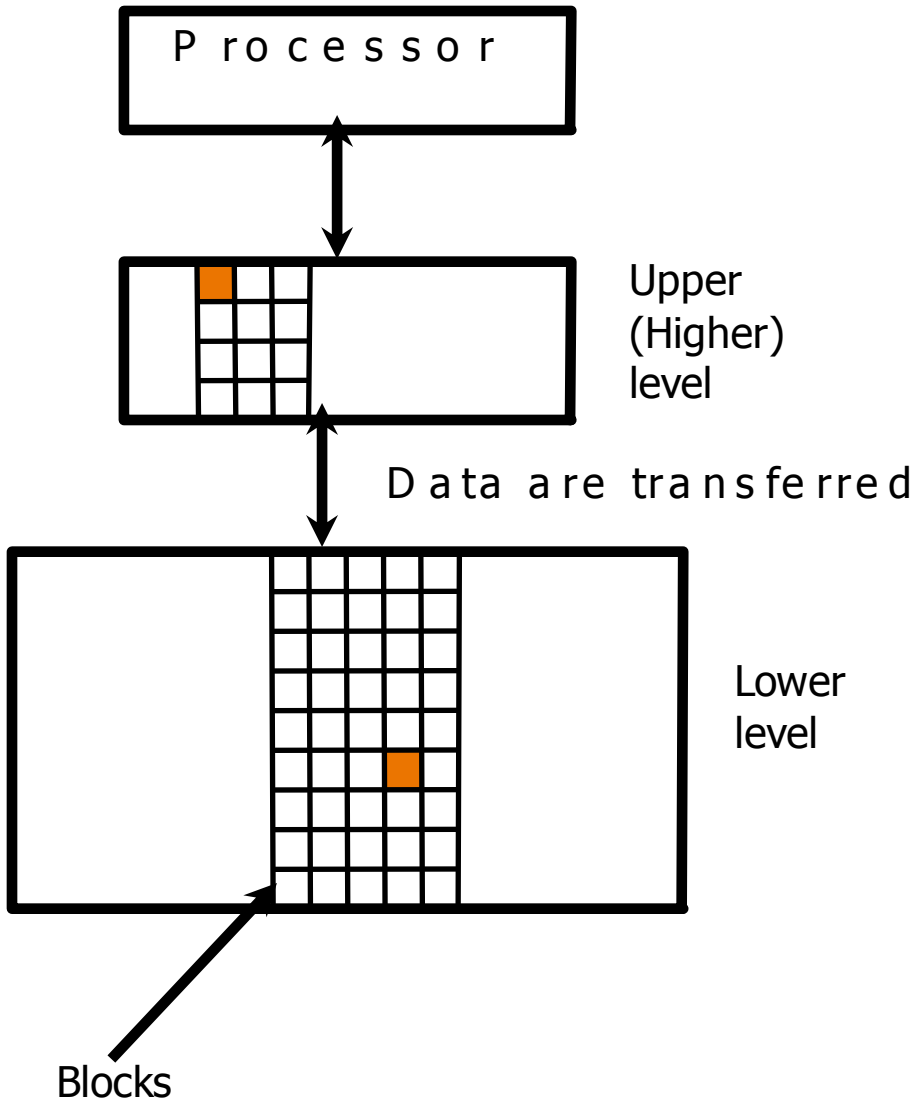
Current technology	Speed	Size	Cost $/bit
Cache (SRAM)	Fastest	Smallest	Highest
Main (DRAM)			
Secondary (Magnetic)	Slowest	Biggest	Lowest

Memory Hierarchy

- Put faster memory closer to CPU
- Goal:
  - Present the user with as much memory as available
  - Provide access at the speed offered by the fastest memory
  - Data could be copied between adjacent levels only
- Result:
  - User thinks that memory is large
  - Memory access is fast as fastest memory
- Illusion
  - CPU access time determined by level 1
  - CPU has memory as large as level n

Memory Hierarchy

- Every pair of levels in the memory hierarchy can be thought of as having an upper & lower level
- Within each level, block is the unit of information
- Transfer is performed block-wise



Locality Principle

- Temporal locality (Time):
  - Tendency to reuse recently accessed data
    => Keep most recent data items closer to the processor
- Spatial locality (Space):
  - Tendency to reference data items that are close to other recently accessed data items
    => Move blocks of contiguous words to the upper level

Locality Principle

- We take advantage by implementing the memory of a computer as a memory hierarchy
- Move blocks of multiple contiguous words in memory to upper levels of hierarchy
- Accesses that miss, go to lower levels
- If the hit rate is high enough, we have an effective memory hierarchy

Fact: Program Code Have Locality

- From program structure
  - Most programs contain loops
    => Instructions & data are likely to be accessed repeatedly
    => Temporal locality
  - Elements of an array or record are stored together
    => We usually access the elements of related structure simultaneously
    => Spatial locality

Terminology

- Block:
  - Minimum unit of data that can be either present or not present in a memory level
- Hit:
  - When the requested data is found in the upper level
- Hit rate (ratio):
  - The fraction of memory accesses found in the upper level
  - Used as a measure of performance of the memory hierarchy
  - The larger the better
- Hit time:
  - Time to access the upper level of memory hierarchy
  - Includes time to determine whether an access is a hit or miss

Terminology

- Miss:
  - Data requested is not found in the upper level
  - Data need to be read from lower level & stored into upper level
- Miss rate (ratio):
  - 1 - hit ratio
  - The fraction of memory accesses not found in the upper level
  - The lower the better
- Miss penalty:
  - Time to
    - replace a block in the upper level with the corresponding block from lower level +
    - deliver this block to the processor

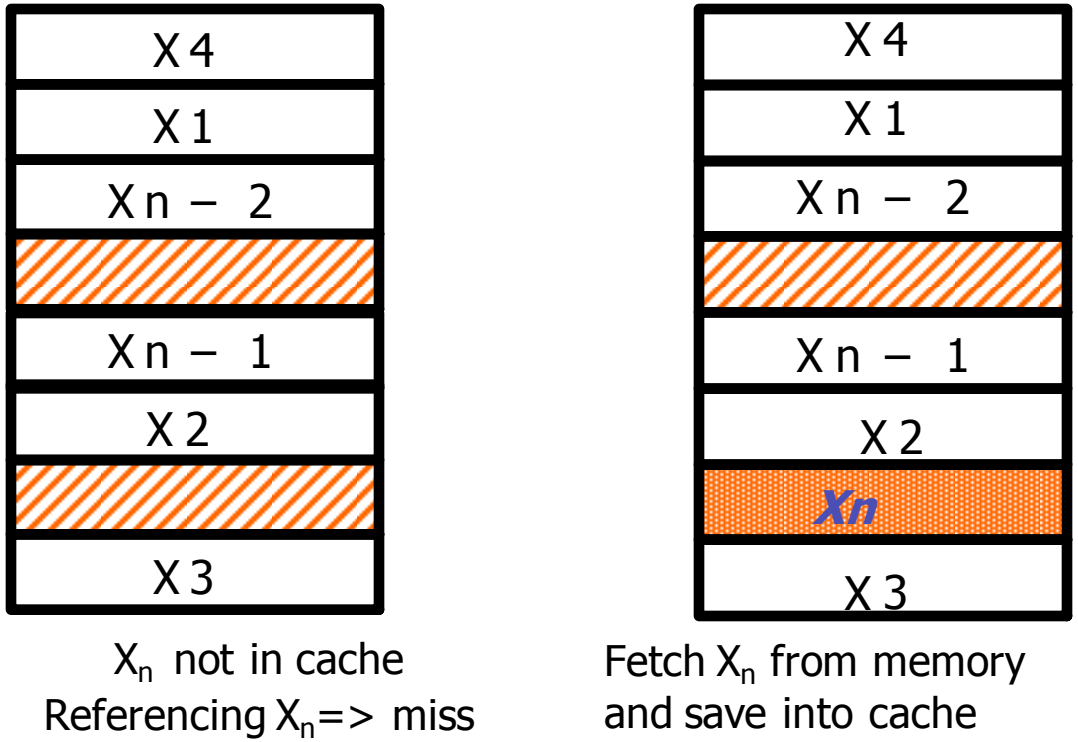
Cache

- What is a cache?
  - Storage managed to take advantage of locality principle
  - Webster's Universal College Dictionary, 1997

" *1. A hiding place for ammunition, food, treasure, etc.*
2. A place of computer HW or section of RAM dedicated to selectively storing and speeding access to frequently used program commands or data"

Cache

- Example:



X_n not in cache
Referencing X_n => miss

Fetch X_n from memory
and save into cache

- Issues:
  - How do we know if a data item is in the cache?
  - If it is, how do we find it?

Average memory access time

- Average memory access time =

  Hit time + Miss time (Mem stalls / access) =

  Hit rate * Hit time + Miss rate * Miss penalty

(For Instruction access and Data access)

% instructions * (Hit time + instruction miss rate*miss penalty)

+

% data * (Hit time + data miss rate*miss penalty)

Direct-Mapped Cache

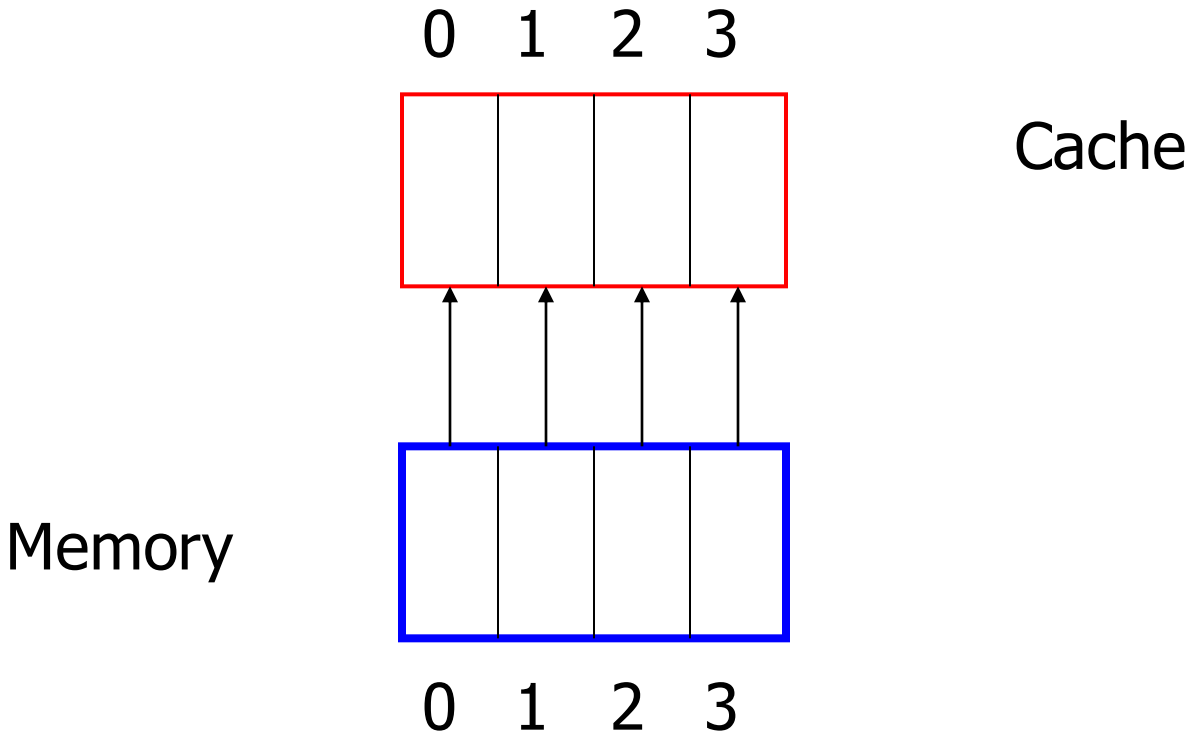
- Each memory location maps to exactly one location in the cache
- One-to-many mapping
- Formula used:
  (block address) modulo (Number of cache blocks in the cache)
- Part of the memory address is used as tag
- Several items at the lower level share locations in the upper level

Direct-Mapped Cache

- Example:
  - Block size is one word of data
- Problem:
  - If two different memory addresses that map to the same location are accessed interchangeably, they will pushing each other
- Possible solution:
  - Use 2 caches in parallel, looking up memory locations in both

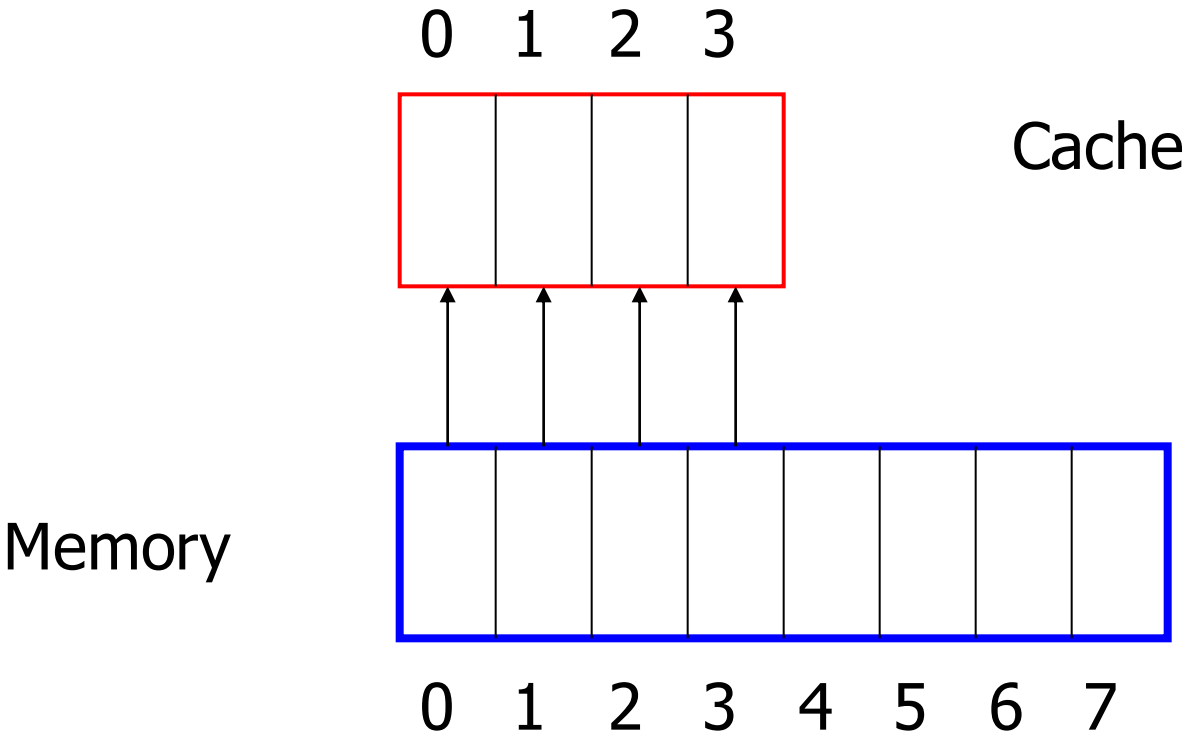
Direct-Mapped Cache

- Example:
  - Block size is one word of data



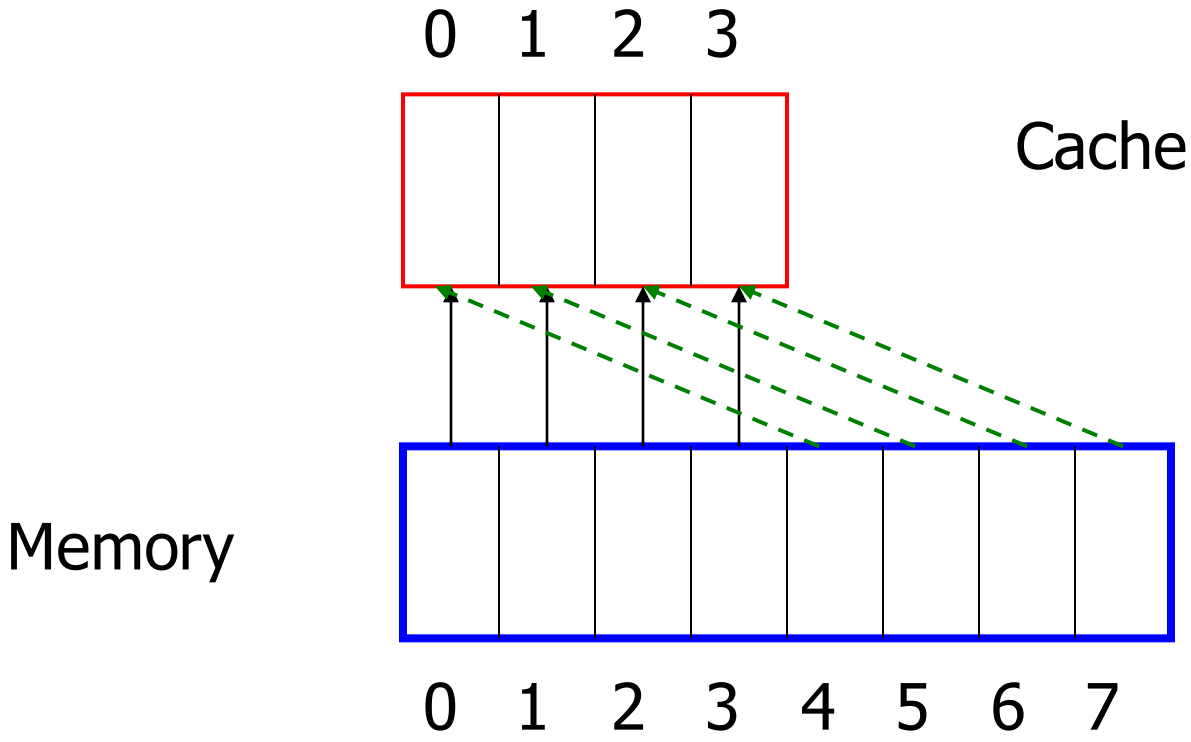
Direct-Mapped Cache

- Example:
  - Block size is one word of data



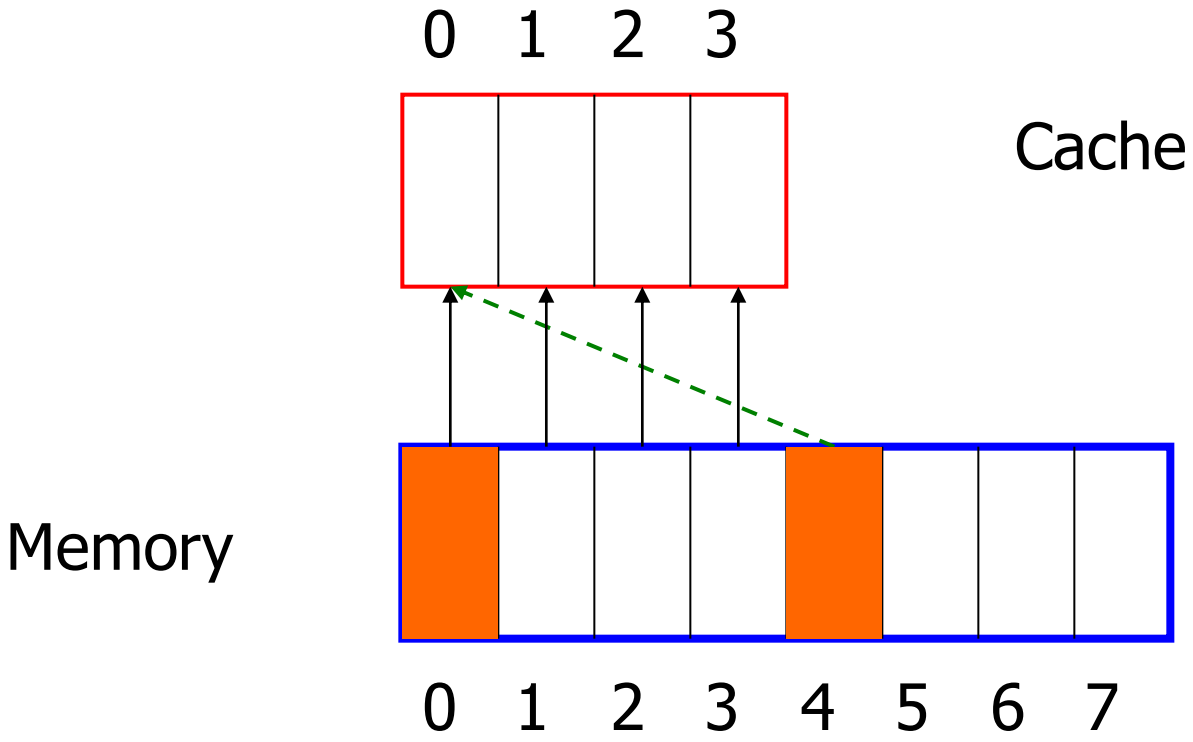
Direct-Mapped Cache

- Example:
  - Block size is one word of data



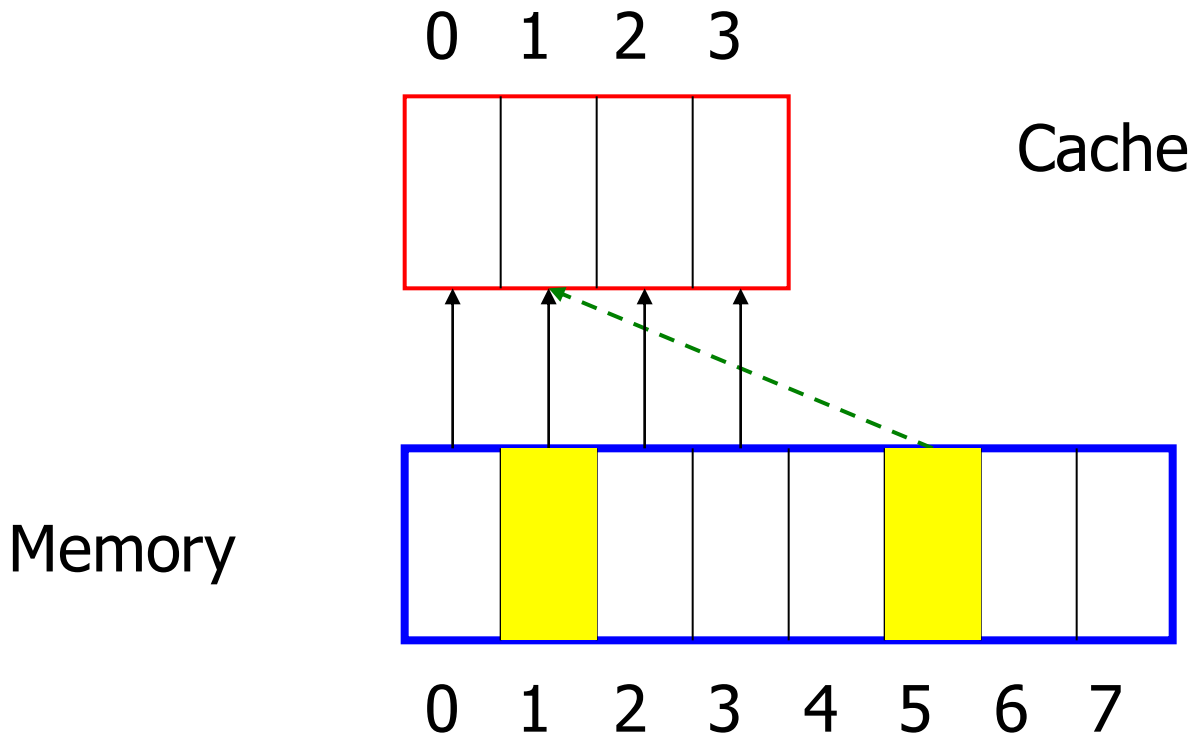
Direct-Mapped Cache

- Example:
  - Block size is one word of data



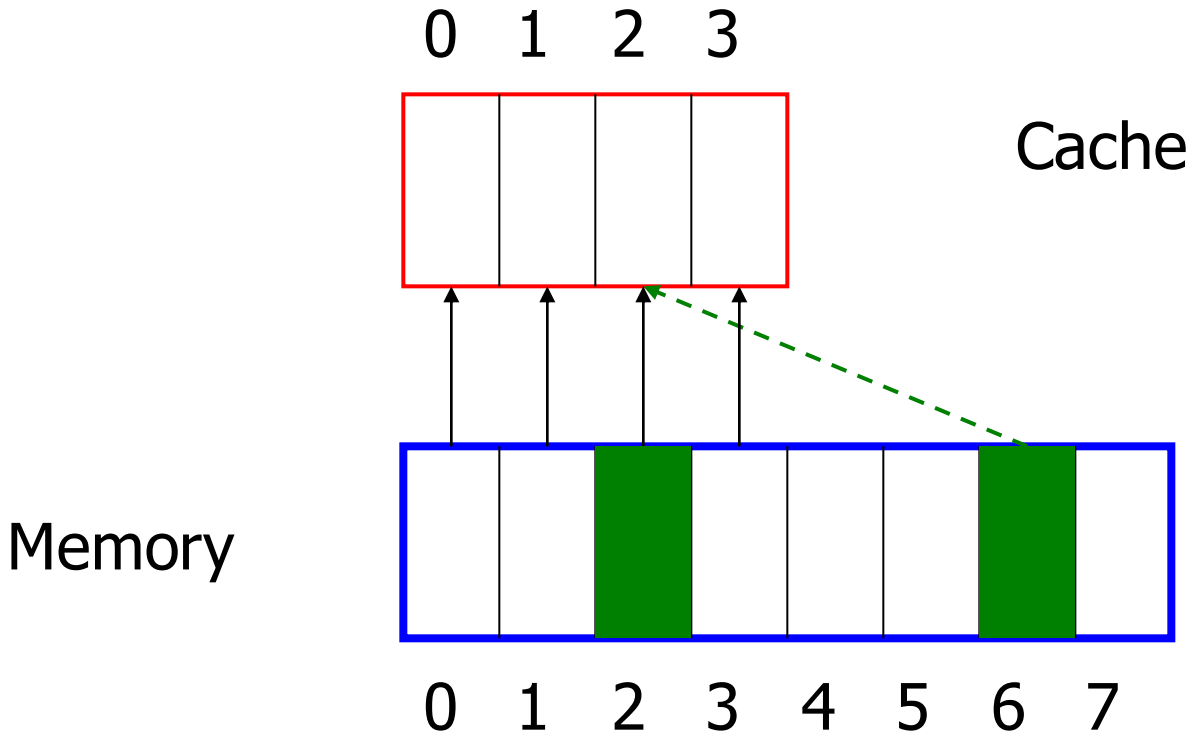
Direct-Mapped Cache

- Example:
  - Block size is one word of data



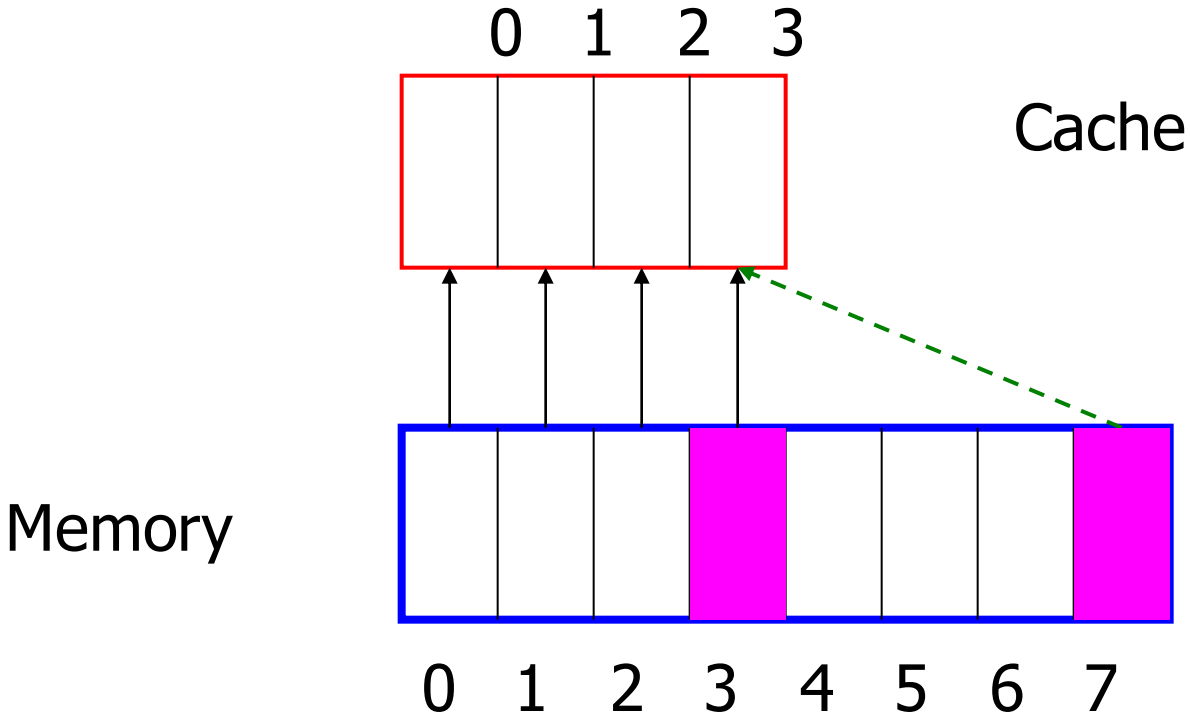
Direct-Mapped Cache

- Example:
  - Block size is one word of data



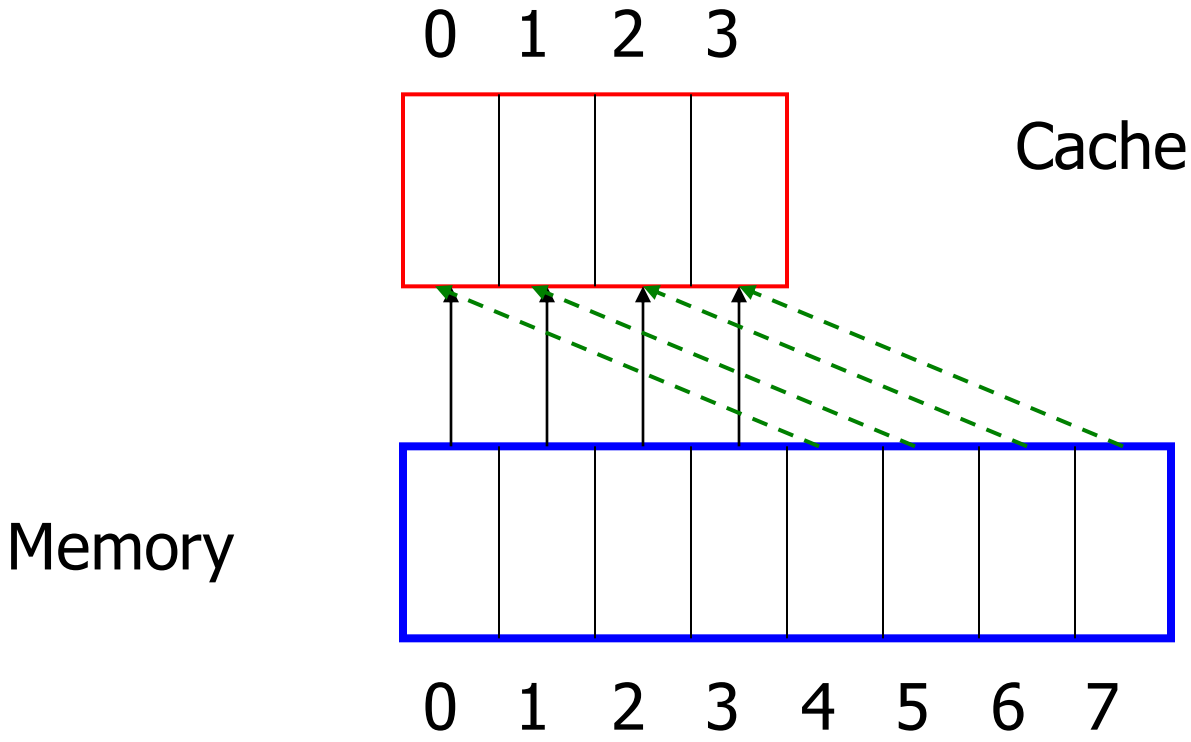
Direct-Mapped Cache

- Example:
  - Block size is one word of data



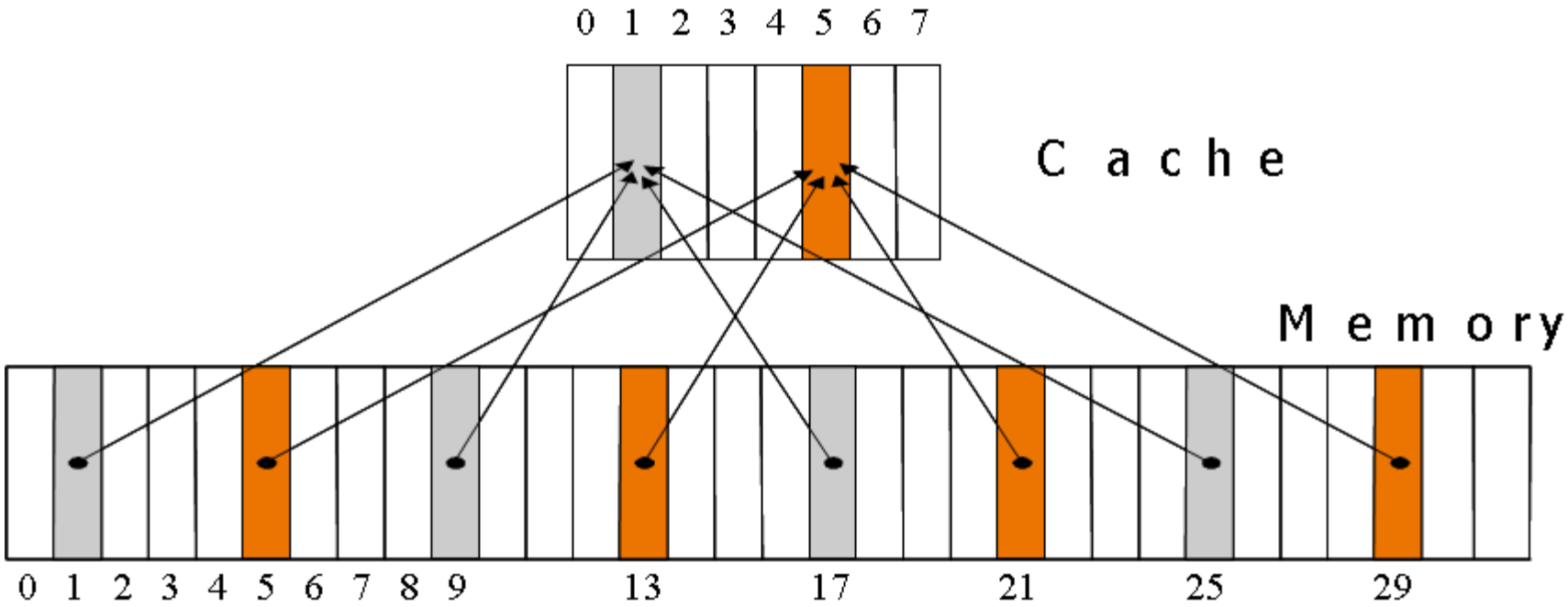
Direct-Mapped Cache

- Example:
  - Block size is one word of data



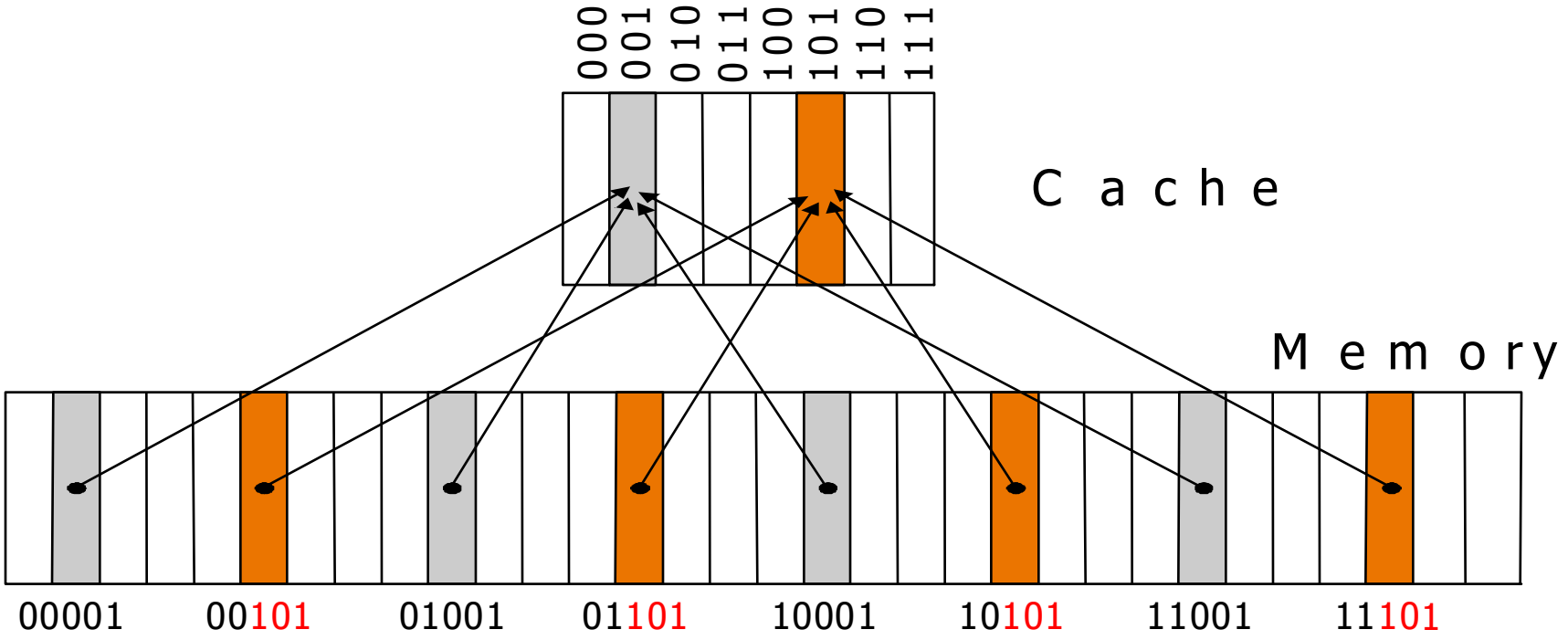
Direct Mapped Cache

- Mapping:
  - Block size is one word of data
  - Cache has 8 entries
  - Address is modulo the number of blocks in the cache



Direct Mapped Cache

- Mapping:
  - Block size is one word of data
  - Cache has 8 entries
  - Cache index: Use the rightmost 3-digit to map the address
  - Address is modulo the number of blocks in the cache
- Cache accessed directly using low-order bits



Direct Mapped Cache

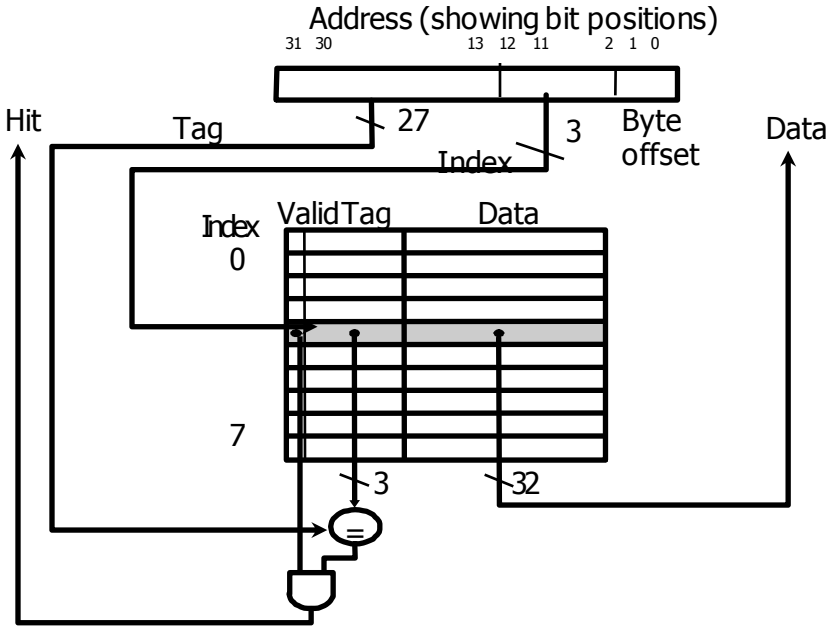
- Problem:
  - How do we know whether the requested word is in the cache?
- Solution: Tag field
  - Added to data word in cache
  - Contain address information to where the data belongs
  - Need only contain the upper portion of the address (not used as index to the cache)

Direct Mapped Cache

- Problem:
  - How do we know that the block has the valid information?
- Solution: Valid bit
  - Added to data word in cache
  - 1 means data is valid (valid address)
  - 0 means data is invalid (Invalid address)

Direct Mapped Cache

- Example:
  - Cache has 8 blocks
  - 1 block = 1 word (4 bytes)
  - 32-bit address
  - Byte offset:
    - The first 2 bits give the byte number (byte offset)
  - Cache index:
    - Next Low-order 3 bits of the address will give the block number(cache index)
  - The rest of the bits give the tag (address of block in higher level memory)
  - Both data & tag are stored in each entry of the cache
  - Some other bits can be stored
    - Valid bit
    - …



Direct Mapped Cache

- Example
  - Draw the cache after inserting the following blocks

    10110, 11010, 10000, 00011, 10010
- Initial Cache

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

Direct Mapped Cache

- After 10110

Index	V	Tag	Data
000	N		
001	N		
010	n		
011	N		
100	N		
101	N		
110	y	10	Memory(10110)
111	N		

- After 11010

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Memory(11010)
011	N		
100	N		
101	N		
110	y	10	Memory(10110)
111	N		

Direct Mapped Cache

- After 10000

Index	V	Tag	Data
000	Y	10	Memory(10000)
001	N		
010	Y	11	Memory(11010)
011	N		
100	N		
101	N		
110	y	10	Memory(10110)
111	N		

- After 00011

Index	V	Tag	Data
000	Y	10	Memory(10000)
001	N		
010	Y	11	Memory(11010)
011	Y	00	Memory(00011)
100	N		
101	N		
110	y	10	Memory(10110)
111	N		

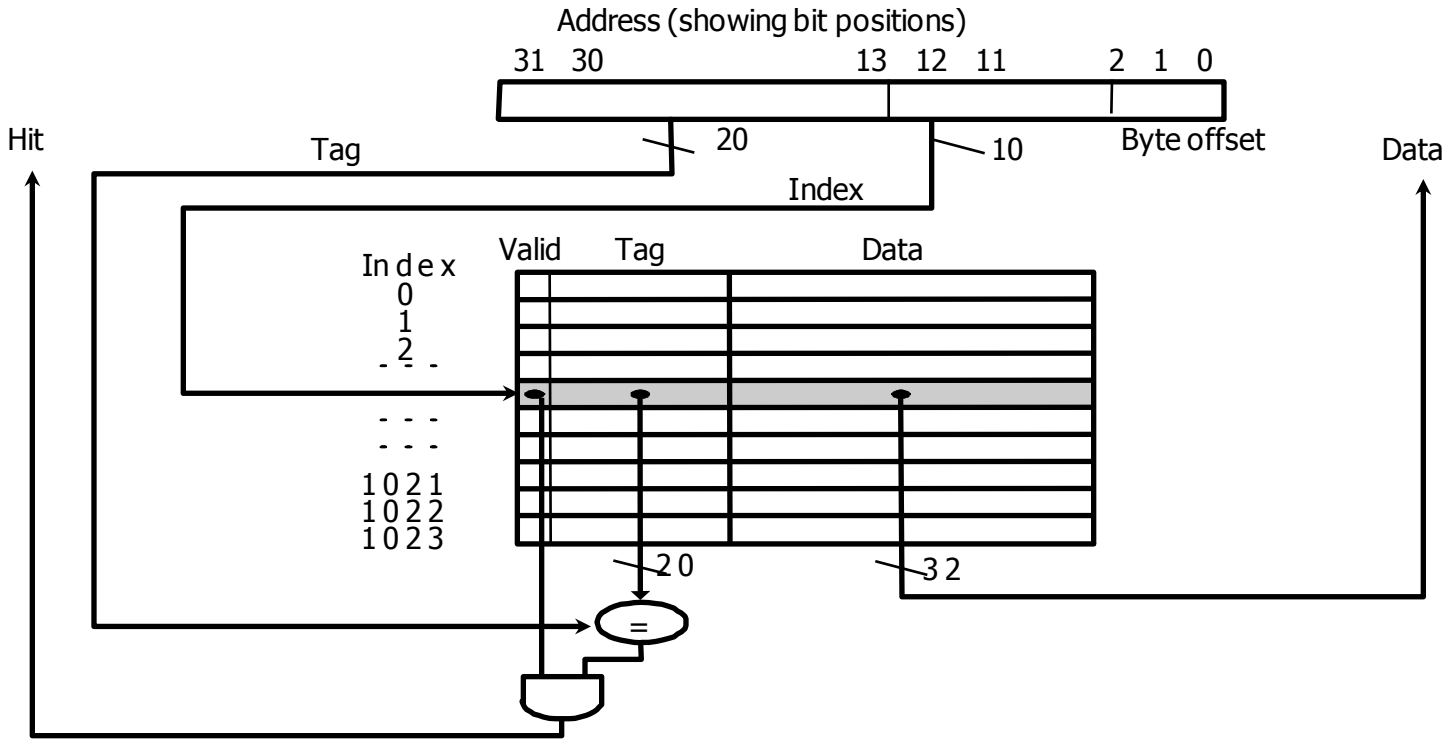
Direct Mapped Cache

- After 10010

Index	V	Tag	Data
000	Y	10	Memory(10000)
001	N		
010	Y	10	Memory(10010)
011	Y	00	Memory(00011)
100	N		
101	N		
110	y	10	Memory(10110)
111	N		

Direct Mapped Cache for MIPS

- Cache size 2^{10} = 1024
- Index bits = 10
- Tag = 32 – 10 – 2 = 20
- tag from cache compared against upper portion of address
- If match Þ Entry in cache corresponds to requested address
- If valid bit = 1 Þ Request hits Þ word supplied to processor
- If valid bit = 0 Þ Request misses Þ bring word into cache



Hits vs. Misses

- Read hits
  - This is what we want!
- Read misses
  - Stall the CPU,
  - Fetch block from memory
  - Deliver to cache
  - Restart
- Write hits:
  - Can replace data in cache and memory (write-through)
  - Write data only into the cache (write-back the cache later)
- Write misses:
  - Read the entire block into the cache, then write the word

Handling Cache Misses

- The action taken for cache miss depends on whether the action is to access instruction or data
- Data Access
  - Stall the processor until the memory responds with the data
- Instruction access:
  - The contents of the instruction register are invalid
    - Get the instruction from the lower-level memory into the cache
    - Decrement PC contents by 4 to get the address of the current instruction, since the PC is already incremented at the first clock cycle

Handling Write Misses

- Two approaches
  - Write through
  - Write back

Write-Through Approach

- Idea:
  - Always write data into both memory & cache
- Steps:
  1. Index the cache (Bits 15-2 of address)
  2. Write the tag, data, & valid bit into cache
  3. Write the word to main memory using the entire address
- Advantages
  - Simple algorithm
- Disadvantages
  - Poor performance
    - Writing into main memory slows down the machine

Write-Back Approach

- Handles writes by updating values only to the block in the cache, then writing the modified block to the lower level of the hierarchy when the block is replaced
- Steps:
  1. Index the cache
  2. Write the tag, data, & valid bit into cache
  3. The modified block is written to the memory only when it is replaced
- Advantages:
  - Improves performance
- Disadvantages:
  - Complex algorithm

Fully Associative Mapped Cache

- Example
  - Draw the cache after inserting the following blocks

    10110, 11010, 10000, 00011, 10010
- Initial Cache

Index	Arrival Time	V	Tag	Data
00	0	N		
01	0	N		
10	0	N		
11	0	N		

Fully Associative Mapped Cache

- After 10110

Index	Arrival Time	V	Tag	Data
00	1	Y	10110	Memory(10110)
01	0	N		
10	0	N		
11	0	N		

- After 11010

Index	Arrival Time	V	Tag	Data
00	1	Y	10110	Memory(10110)
01	2	Y	11010	Memory(11010)
10	0	N		
11	0	N		

Fully Associative Mapped Cache

- After 10000

Index	Arrival Time	V	Tag	Data
00	1	Y	10110	Memory(10110)
01	2	Y	11010	Memory(11010)
10	3	Y	10000	Memory(10000)
11	4	N		

- After 00011

Index	Arrival Time	V	Tag	Data
00	1	Y	10110	Memory(10110)
01	2	Y	11010	Memory(11010)
10	3	Y	10000	Memory(10000)
11	4	Y	00011	Memory(00011)

Fully Associative Mapped Cache

- After 10010

Index	Arrival Time	V	Tag	Data
00	5	Y	10010	Memory(10010)
01	2	Y	11010	Memory(11010)
10	3	Y	10000	Memory(10000)
11	4	Y	00011	Memory(00011)

Example:

- For a direct-mapped cache with **4K** words, block is **16** words and the main memory contains **8 M** words.
  - Calculate the following:
    - The address bus of the Main memory.
    - The address bus of the Cache memory.
    - The size of the tag field.
    - The size of the word field.
    - The size of the block field.
    - The number of the blocks in the main memory that are mapped to the same block in the cache memory.
    - The address of the cache that contains the memory address: **0002135A**$_h$

Virtual Memory

- Similar to caching, but between main memory & secondary storage instead of RAM & CPU
- Main memory acts as a "cache" for the secondary storage
- Only a portion of the program can be active in main memory, the rest will be in secondary storage
- Multiple program can share the same memory
- Programs sharing the memory change dynamically

Virtual Memory

- Advantages
  - Allows efficient & safe sharing of memory among multiple programs
  - Larger address space without extra programming requirement
    - Illusion of having more physical memory
- Disadvantages
  - Overhead for
    - Memory management
    - Handling page faults
    - Program relocation
      - Converting virtual address space into physical address
  - Programs need to be protected from each other

Virtual Memory