Memory

ICS 233

Computer Architecture and Assembly Language

Dr. Aiman El-Maleh

College of Computer Sciences and Engineering

King Fahd University of Petroleum and Minerals

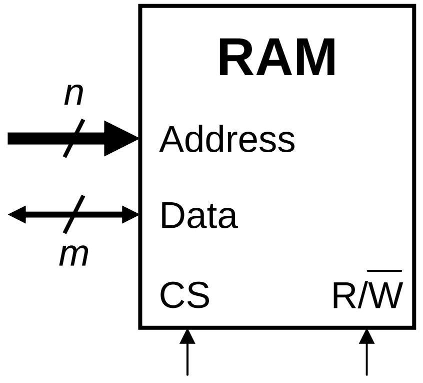
[Adapted from slides of Dr. M. Mudawar, ICS 233, KFUPM]

Outline

- ❖ Random Access Memory and its Structure
- ❖ Memory Hierarchy and the need for Cache Memory
- ❖ The Basics of Caches
- ❖ Cache Performance and Memory Stall Cycles
- ❖ Improving Cache Performance
- ❖ Multilevel Caches

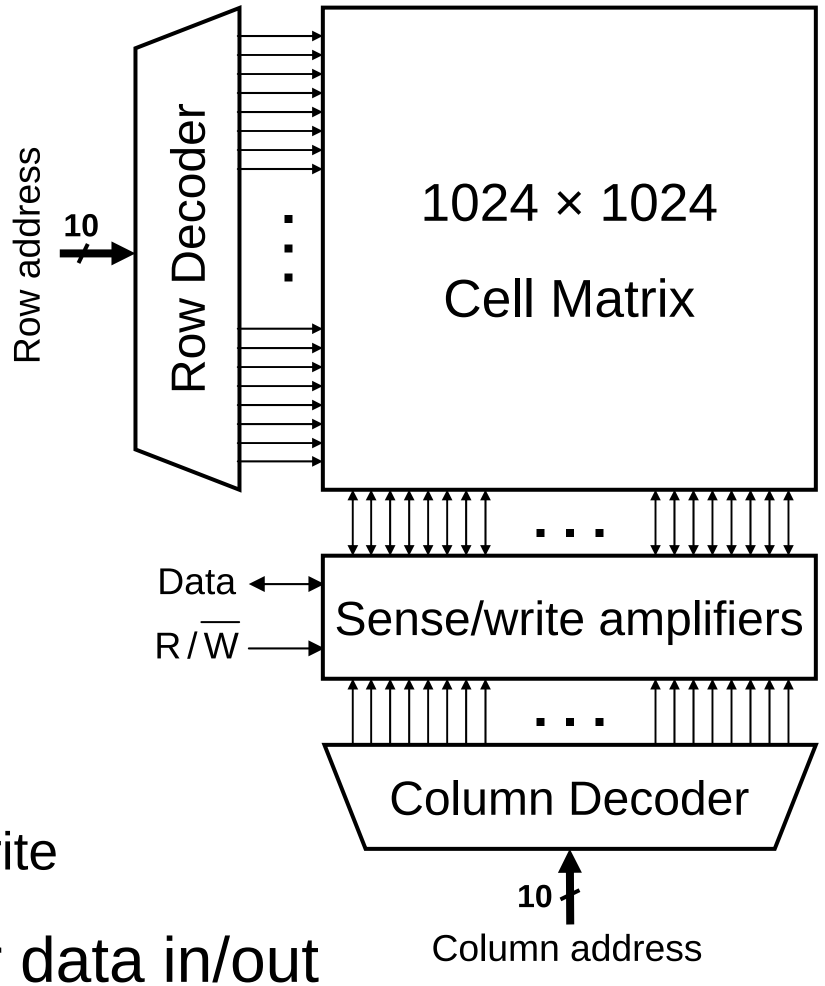
Random Access Memory

- ❖ Large arrays of storage cells
- ❖ Volatile memory
  - ✧ Hold the stored data as long as it is powered on
- ❖ Random Access
  - ✧ Access time is practically the same to any data on a RAM chip
- ❖ Chip Select (CS) control signal
  - ✧ Select RAM chip to read/write
- ❖ Read/Write ($R/\overline{W}$) control signal
  - ✧ Specifies memory operation
- ❖ $2^n \times m$ RAM chip: *n*-bit address and *m*-bit data



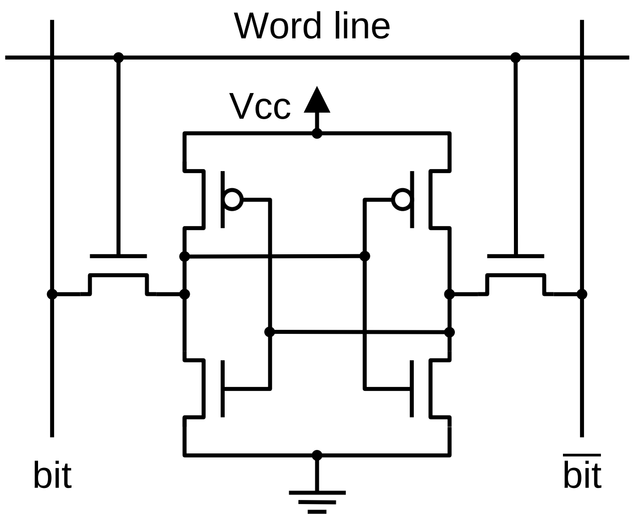
Typical Memory Structure

- Row decoder
  - Select row to read/write
- Column decoder
  - Select column to read/write
- Cell Matrix
  - 2D array of tiny memory cells
- Sense/Write amplifiers
  - Sense & amplify data on read
  - Drive bit line with data in on write
- Same data lines are used for data in/out



Static RAM Storage Cell

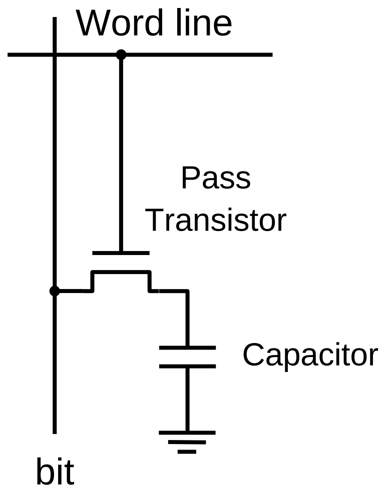
- ❖ Static RAM (SRAM): fast but expensive RAM
- ❖ 6-Transistor cell with no static current
- ❖ Typically used for caches
- ❖ Provides fast access time
- ❖ Cell Implementation:
  - ✧ Cross-coupled inverters store bit
  - ✧ Two pass transistors
  - ✧ Row decoder selects the word line
  - ✧ Pass transistors enable the cell to be read and written



Typical SRAM cell

Dynamic RAM Storage Cell

- ❖ Dynamic RAM (DRAM): slow, cheap, and dense memory
- ❖ Typical choice for main memory
- ❖ Cell Implementation:
  - ✧ 1-Transistor cell (pass transistor)
  - ✧ Trench capacitor (stores bit)
- ❖ Bit is stored as a charge on capacitor
- ❖ Must be refreshed periodically
  - ✧ Because of leakage of charge from tiny capacitor
- ❖ Refreshing for all memory rows
  - ✧ Reading each row and writing it back to restore the charge



Typical DRAM cell

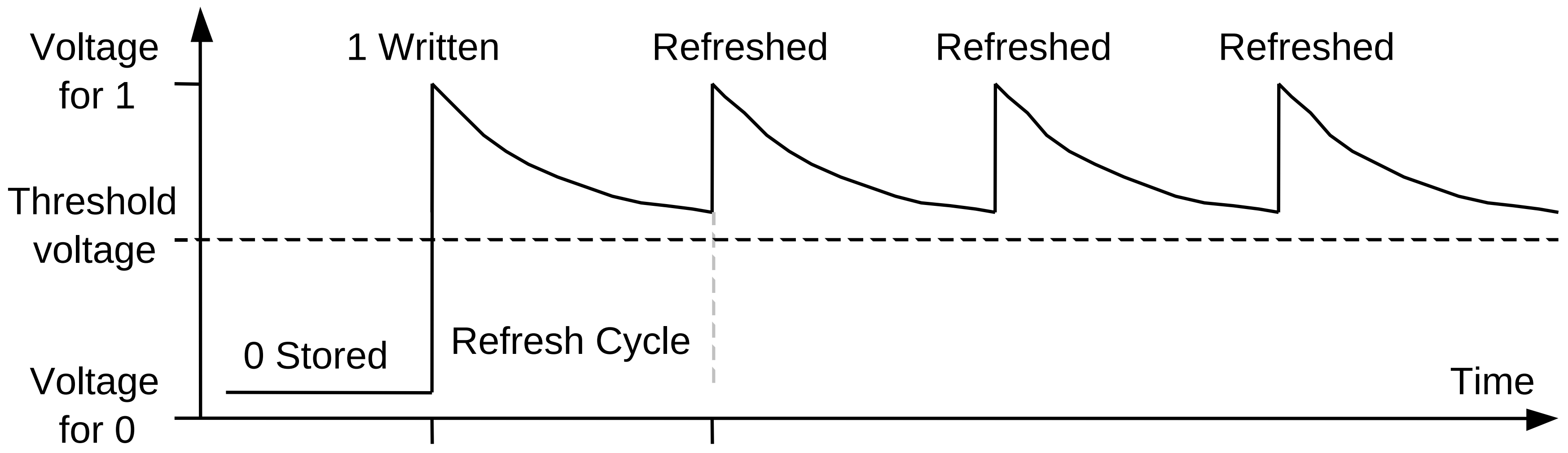
DRAM Refresh Cycles

❖ Refresh cycle is about tens of milliseconds

❖ Refreshing is done for the entire memory

❖ Each row is read and written back to restore the charge

❖ Some of the memory bandwidth is lost to refresh cycles



Loss of Bandwidth to Refresh Cycles

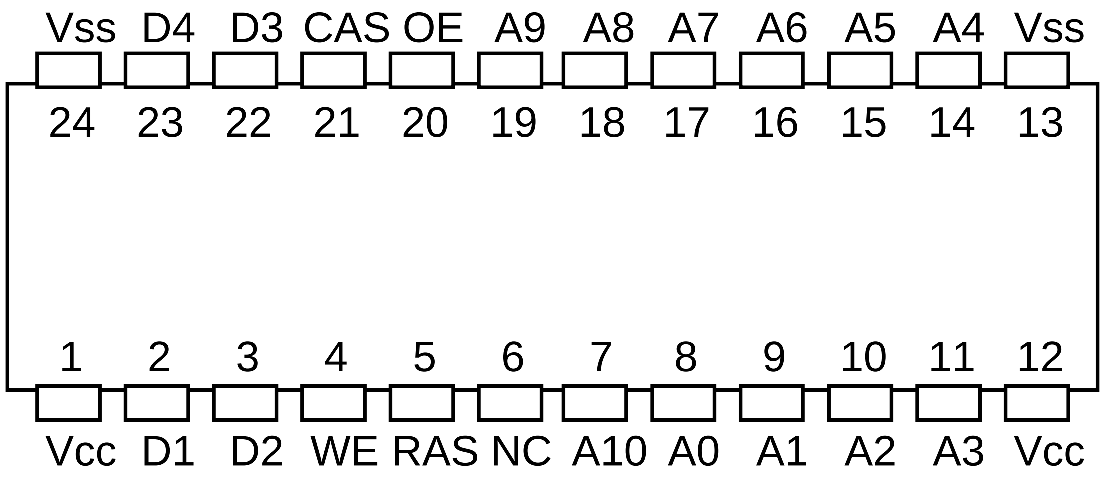
- Example:
  - A 256 Mb DRAM chip
  - Organized internally as a 16K × 16K cell matrix
  - Rows must be refreshed at least once every 50 ms
  - Refreshing a row takes 100 ns
  - What fraction of the memory bandwidth is lost to refresh cycles?
- Solution:
  - Refreshing all 16K rows takes: 16 × 1024 × 100 ns = 1.64 ms
  - Loss of 1.64 ms every 50 ms
  - Fraction of lost memory bandwidth = 1.64 / 50 = 3.3%

Typical DRAM Packaging

- 24-pin dual in-line package for 16Mbit = $2^{22} \times 4$ memory
- 22-bit address is divided into
  - 11-bit row address
  - 11-bit column address
  - Interleaved on same address lines

Legend

A*i*	Address bit *i*
CAS	Column address strobe
D*j*	Data bit *j*
NC	No connection
OE	Output enable
RAS	Row address strobe
WE	Write enable



Vss D4 D3 CAS OE A9 A8 A7 A6 A5 A4 Vss
24 23 22 21 20 19 18 17 16 15 14 13
1 2 3 4 5 6 7 8 9 10 11 12
Vcc D1 D2 WE RAS NC A10 A0 A1 A2 A3 Vcc

Trends in DRAM

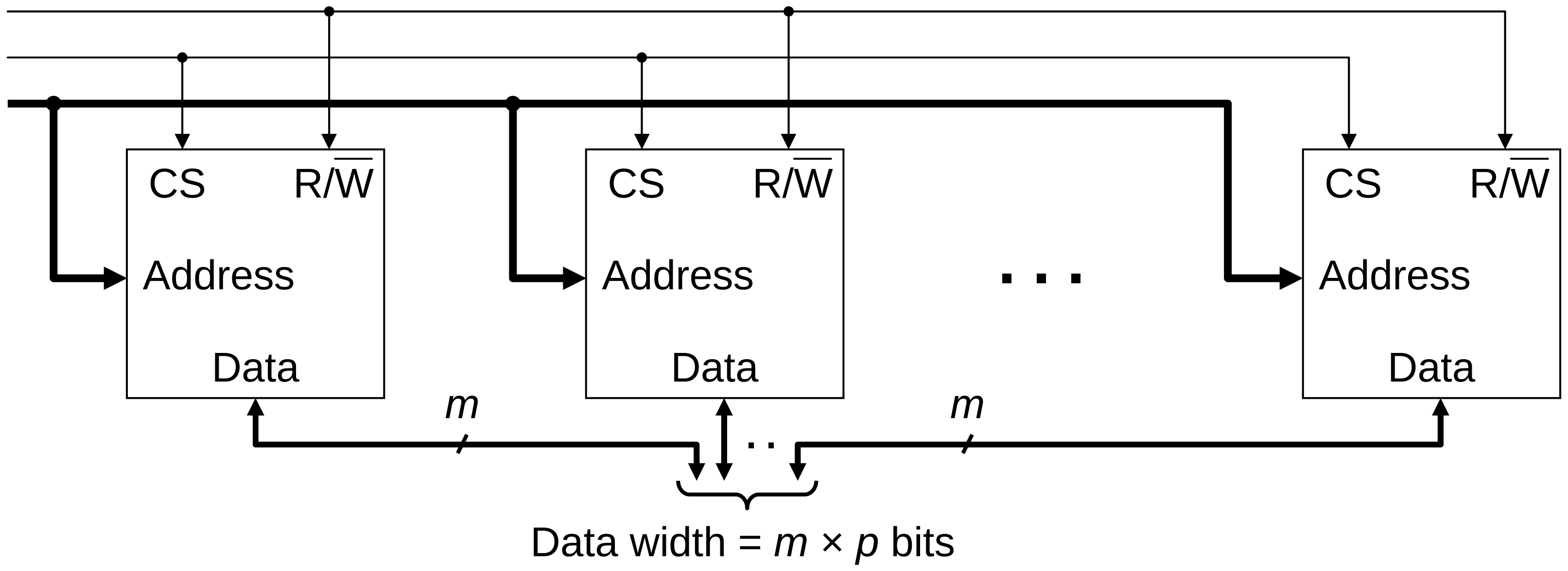
DRAM capacity quadrupled every three years until 1996

After 1996, DRAM capacity doubled every two years

Year introduced	Capacity	Cost per MB	Total access time to a new row	Column access to existing row
1980	64 Kbit	$1500.00	250 ns	150 ns
1983	256 Kbit	$500.00	185 ns	100 ns
1985	1 Mbit	$200.00	135 ns	40 ns
1989	4 Mbit	$50.00	110 ns	40 ns
1992	16 Mbit	$15.00	90 ns	30 ns
1996	64 Mbit	$10.00	60 ns	12 ns
1998	128 Mbit	$4.00	60 ns	10 ns
2000	256 Mbit	$1.00	55 ns	7 ns
2002	512 Mbit	$0.25	50 ns	5 ns
2004	1024 Mbit	$0.10	45 ns	3 ns

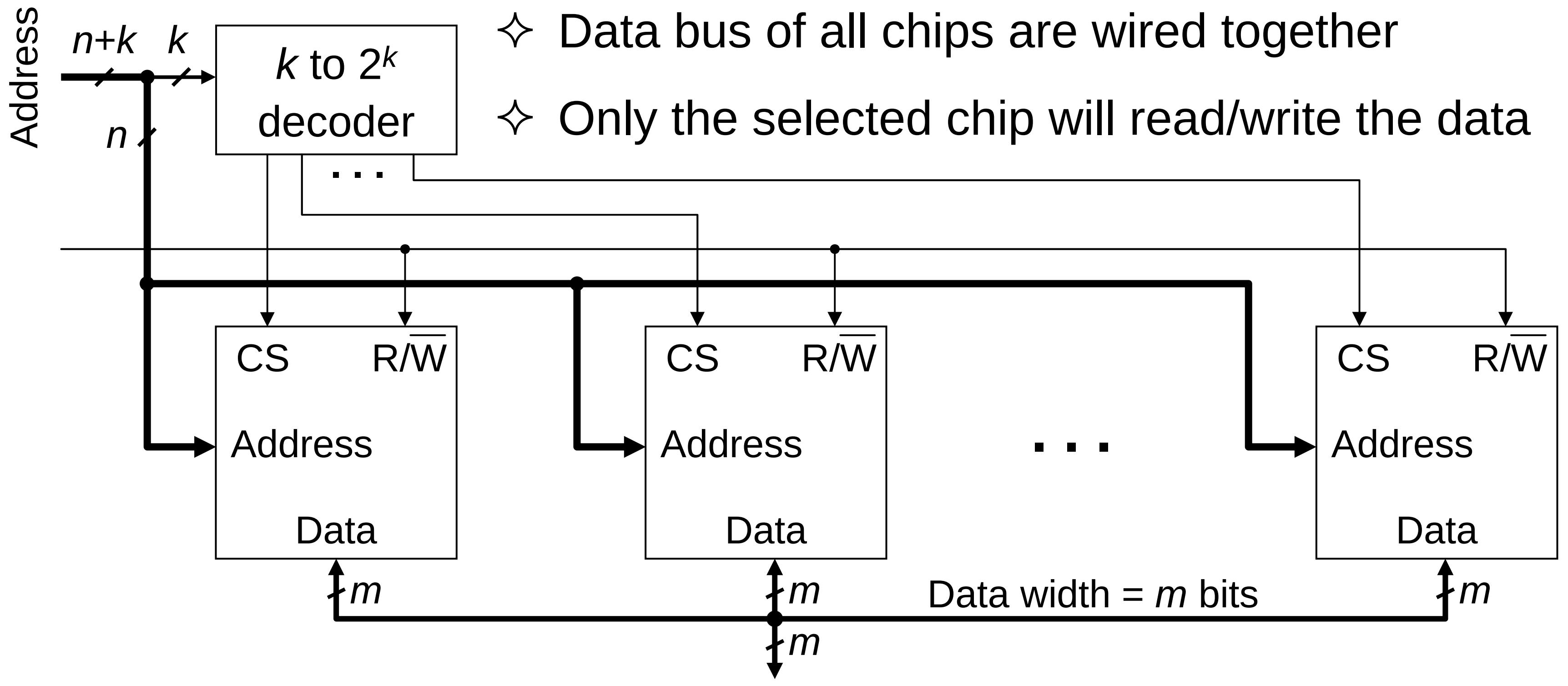
Expanding the Data Bus Width

- ❖ Memory chips typically have a narrow data bus
- ❖ We can expand the data bus width by a factor of *p*
  - ✧ Use *p* RAM chips and feed the same address to all chips
  - ✧ Use the same Chip Select and Read/Write control signals



Increasing Memory Capacity by 2^k

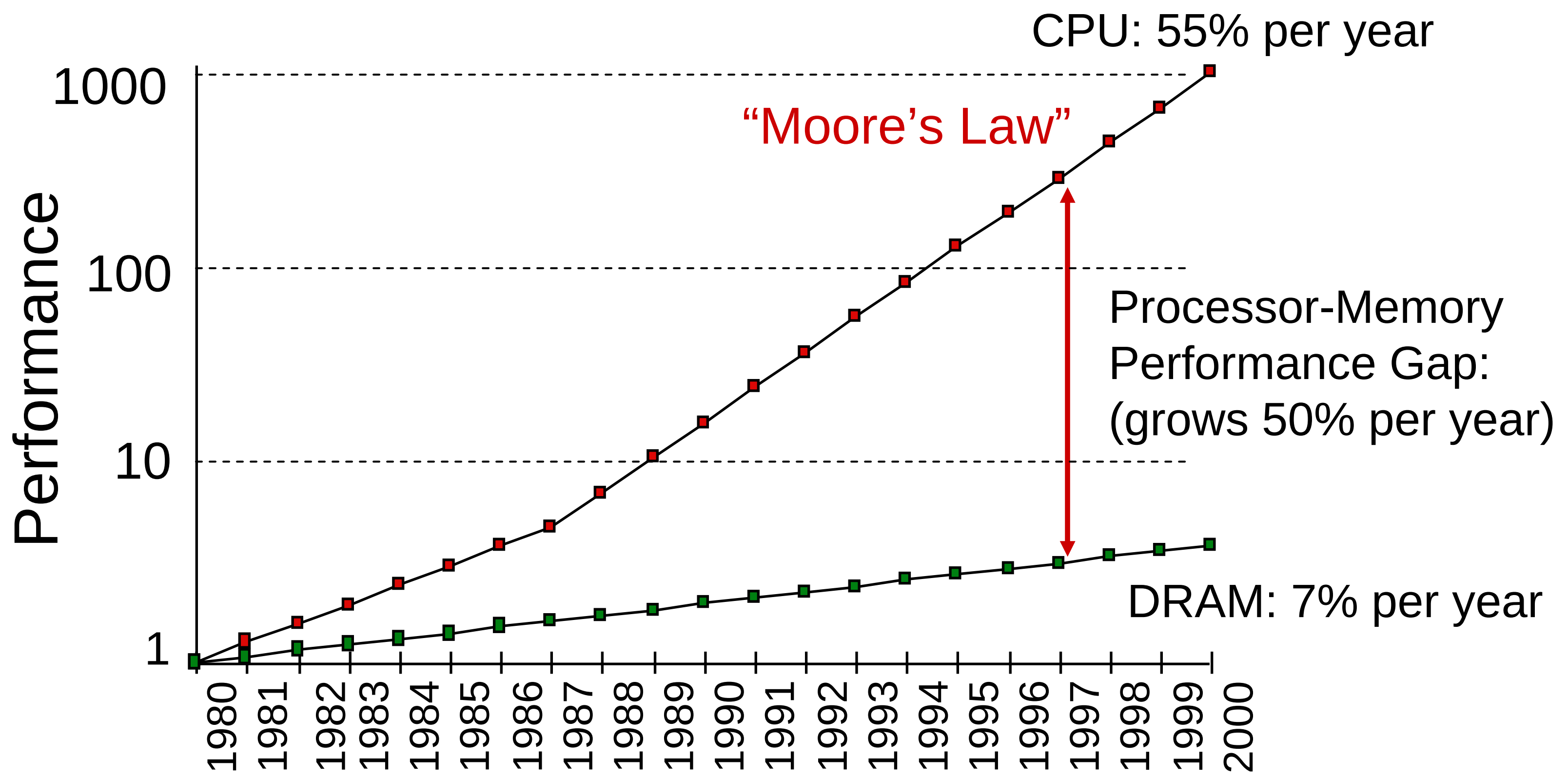
- A k to 2^k decoder is used to select one of the 2^k chips
  - Upper n bits of address is fed to all memory chips
  - Lower k bits of address are decoded to select one of the 2^k chips
  - Data bus of all chips are wired together
  - Only the selected chip will read/write the data



Next . . .

- Random Access Memory and its Structure
- Memory Hierarchy and the need for Cache Memory
- The Basics of Caches
- Cache Performance and Memory Stall Cycles
- Improving Cache Performance
- Multilevel Caches

Processor-Memory Performance Gap



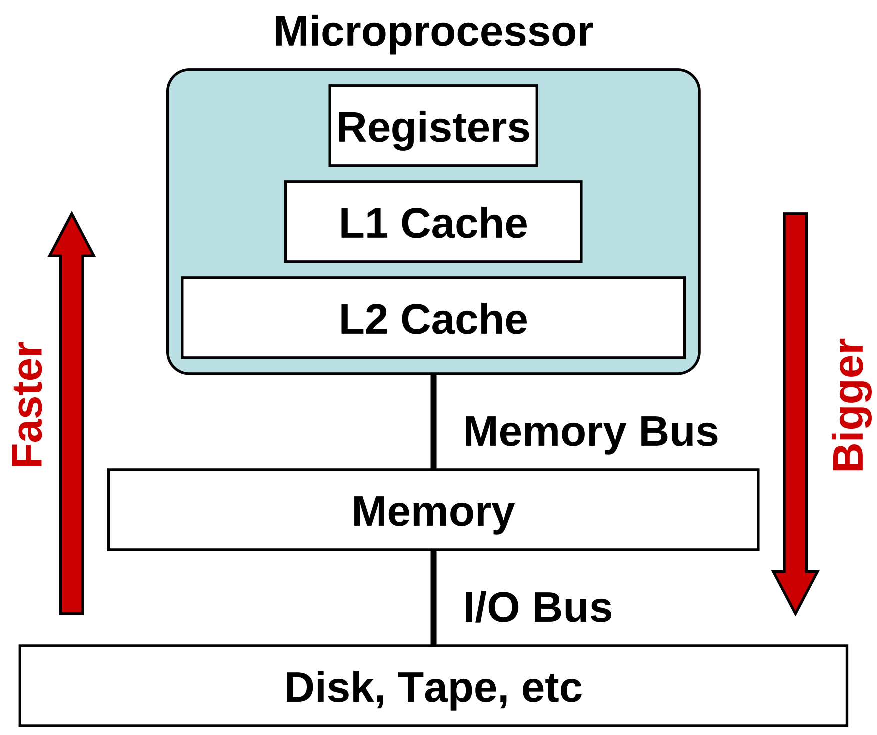
- 1980 – No cache in microprocessor
- 1995 – Two-level cache on microprocessor

The Need for a Memory Hierarchy

- Widening speed gap between CPU and main memory
  - Processor operation takes less than 1 ns
  - Main memory requires more than 50 ns to access
- Each instruction involves at least one memory access
  - One memory access to fetch the instruction
  - A second memory access for load and store instructions
- Memory bandwidth limits the instruction execution rate
- Cache memory can help bridge the CPU-memory gap
- Cache memory is small in size but fast

Typical Memory Hierarchy

- Registers are at the top of the hierarchy
  - Typical size < 1 KB
  - Access time < 0.5 ns
- Level 1 Cache (8 – 64 KB)
  - Access time: 0.5 – 1 ns
- L2 Cache (512KB – 8MB)
  - Access time: 2 – 10 ns
- Main Memory (1 – 2 GB)
  - Access time: 50 – 70 ns
- Disk Storage (> 200 GB)
  - Access time: milliseconds



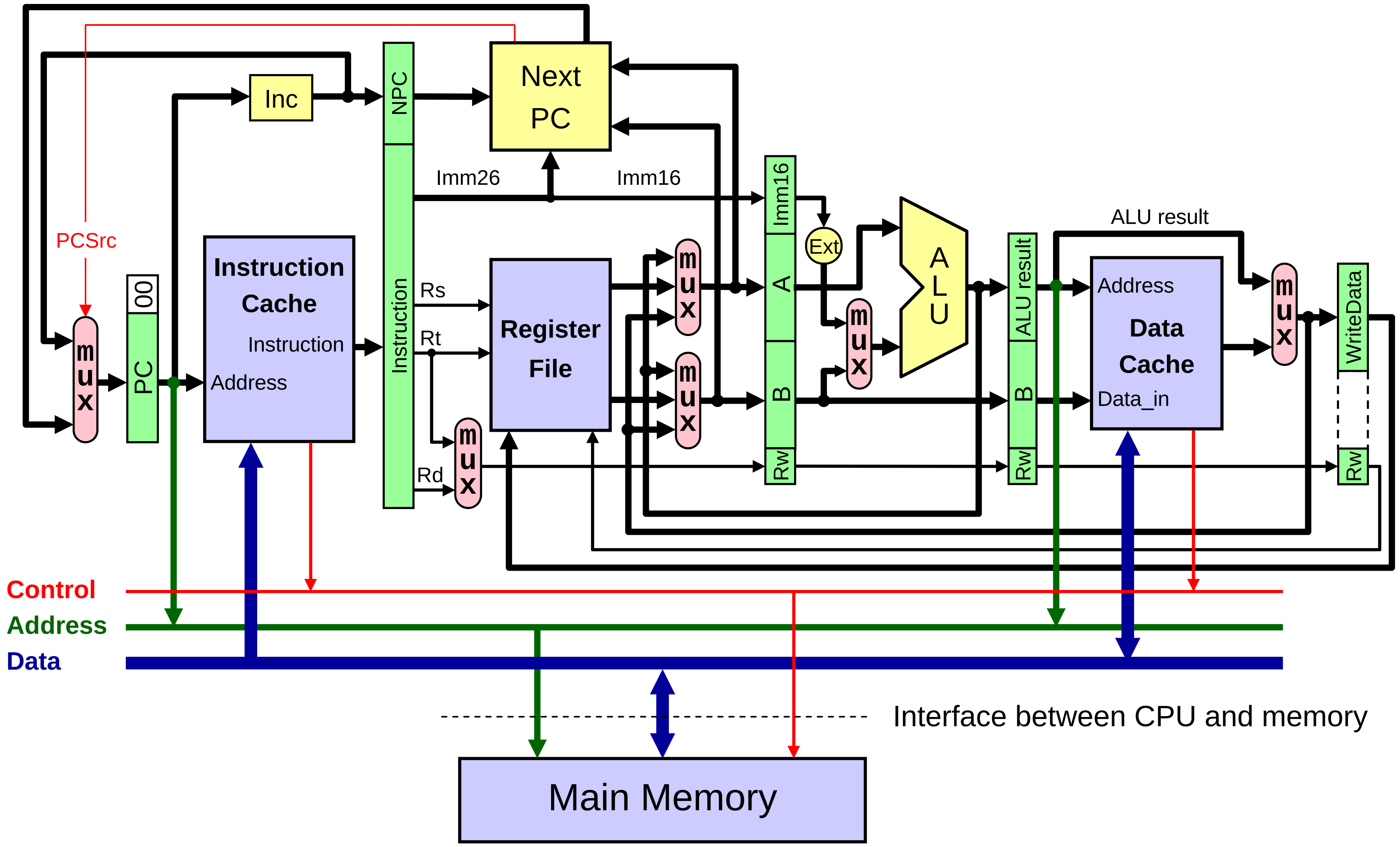
Principle of Locality of Reference

- Programs access small portion of their address space
  - At any time, only a small set of instructions & data is needed
- Temporal Locality (in time)
  - If an item is accessed, probably it will be accessed again soon
  - Same loop instructions are fetched each iteration
  - Same procedure may be called and executed many times
- Spatial Locality (in space)
  - Tendency to access contiguous instructions/data in memory
  - Sequential execution of Instructions
  - Traversing arrays element by element

What is a Cache Memory ?

- Small and fast (SRAM) memory technology
  - Stores the subset of instructions & data currently being accessed
- Used to reduce average access time to memory
- Caches exploit temporal locality by …
  - Keeping recently accessed data closer to the processor
- Caches exploit spatial locality by …
  - Moving blocks consisting of multiple contiguous words
- Goal is to achieve
  - Fast speed of cache memory access
  - Balance the cost of the memory system

Cache Memories in the Datapath



Almost Everything is a Cache !

- In computer architecture, almost everything is a cache!
- Registers: a cache on variables – software managed
- First-level cache: a cache on second-level cache
- Second-level cache: a cache on memory
- Memory: a cache on hard disk
  - Stores recent programs and their data
  - Hard disk can be viewed as an extension to main memory
- Branch target and prediction buffer
  - Cache on branch target and prediction information

Next . . .

- ❖ Random Access Memory and its Structure
- ❖ Memory Hierarchy and the need for Cache Memory
- ❖ The Basics of Caches
- ❖ Cache Performance and Memory Stall Cycles
- ❖ Improving Cache Performance
- ❖ Multilevel Caches

Four Basic Questions on Caches

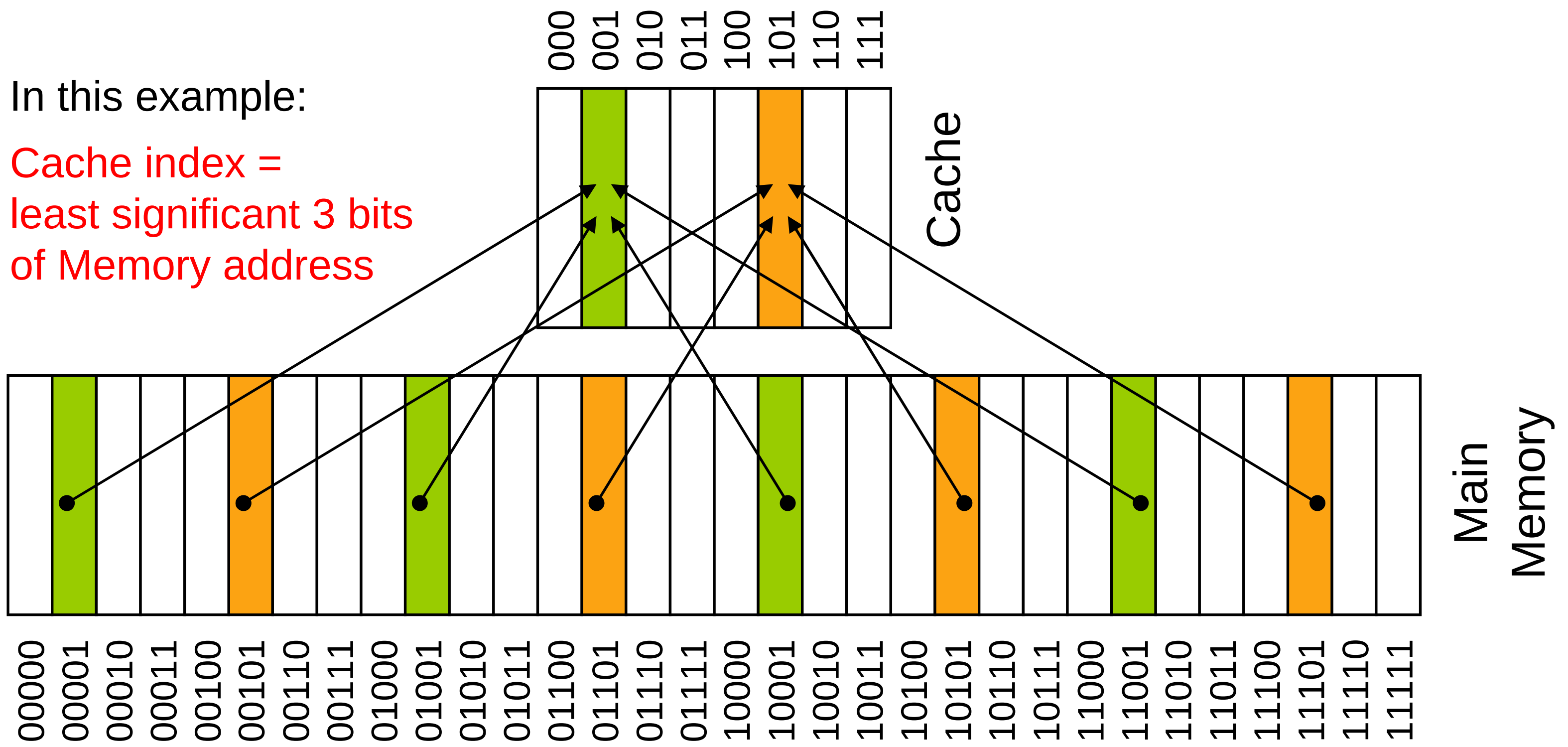
- Q1: Where can a block be placed in a cache?
  - Block placement
  - Direct Mapped, Set Associative, Fully Associative
- Q2: How is a block found in a cache?
  - Block identification
  - Block address, tag, index
- Q3: Which block should be replaced on a miss?
  - Block replacement
  - FIFO, Random, LRU
- Q4: What happens on a write?
  - Write strategy
  - Write Back or Write Through (with Write Buffer)

Block Placement: Direct Mapped

- Block: unit of data transfer between cache and memory
- Direct Mapped Cache:
  - A block can be placed in exactly one location in the cache

In this example:

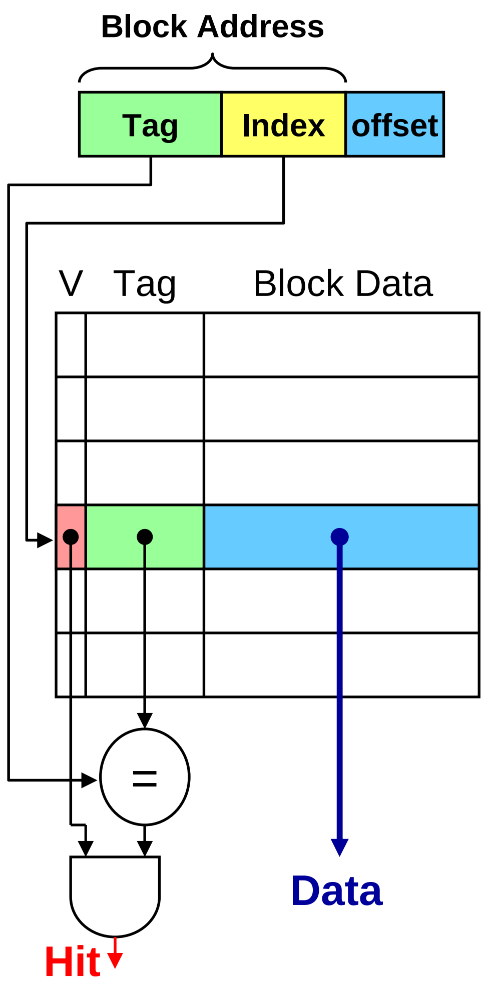
Cache index = least significant 3 bits of Memory address



Direct-Mapped Cache

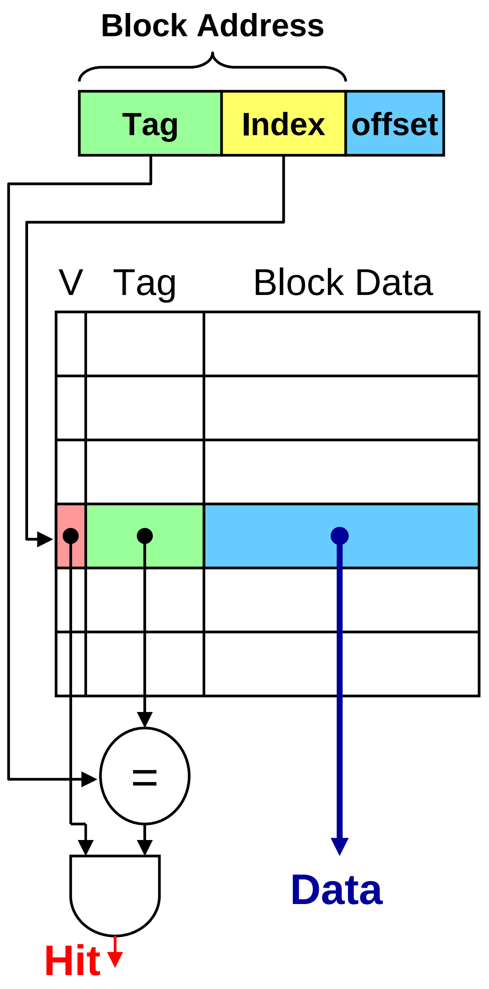
- A memory address is divided into
  - Block address: identifies block in memory
  - Block offset: to access bytes within a block
- A block address is further divided into
  - Index: used for direct cache access
  - Tag: most-significant bits of block address

  Index = Block Address **mod** *Cache Blocks*
- Tag must be stored also inside cache
  - For block identification
- A valid bit is also required to indicate
  - Whether a cache block is valid or not



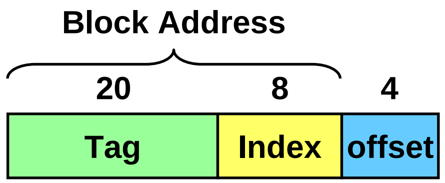
Direct Mapped Cache – cont'd

- Cache hit: block is stored inside cache
  - Index is used to access cache block
  - Address tag is compared against stored tag
  - If equal and cache block is valid then **hit**
  - Otherwise: **cache miss**
- If number of cache blocks is 2^n
  - n bits are used for the cache index
- If number of bytes in a block is 2^b
  - b bits are used for the block offset
- If 32 bits are used for an address
  - $32 - n - b$ bits are used for the tag
- Cache data size = 2^{n+b} bytes



Mapping an Address to a Cache Block

- Example
  - Consider a direct-mapped cache with 256 blocks
  - Block size = 16 bytes
  - Compute tag, index, and byte offset of address: 0x01FFF8AC
- Solution
  - 32-bit address is divided into:

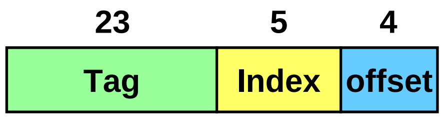


    - 4-bit byte offset field, because block size = 2^4 = 16 bytes
    - 8-bit cache index, because there are 2^8 = 256 blocks in cache
    - 20-bit tag field
  - Byte offset = 0xC = 12 (least significant 4 bits of address)
  - Cache index = 0x8A = 138 (next lower 8 bits of address)
  - Tag = 0x01FFF (upper 20 bits of address)

Example on Cache Placement & Misses

- ❖ Consider a small direct-mapped cache with 32 blocks
  - ✧ Cache is initially empty, Block size = 16 bytes
  - ✧ The following memory addresses (in decimal) are referenced:

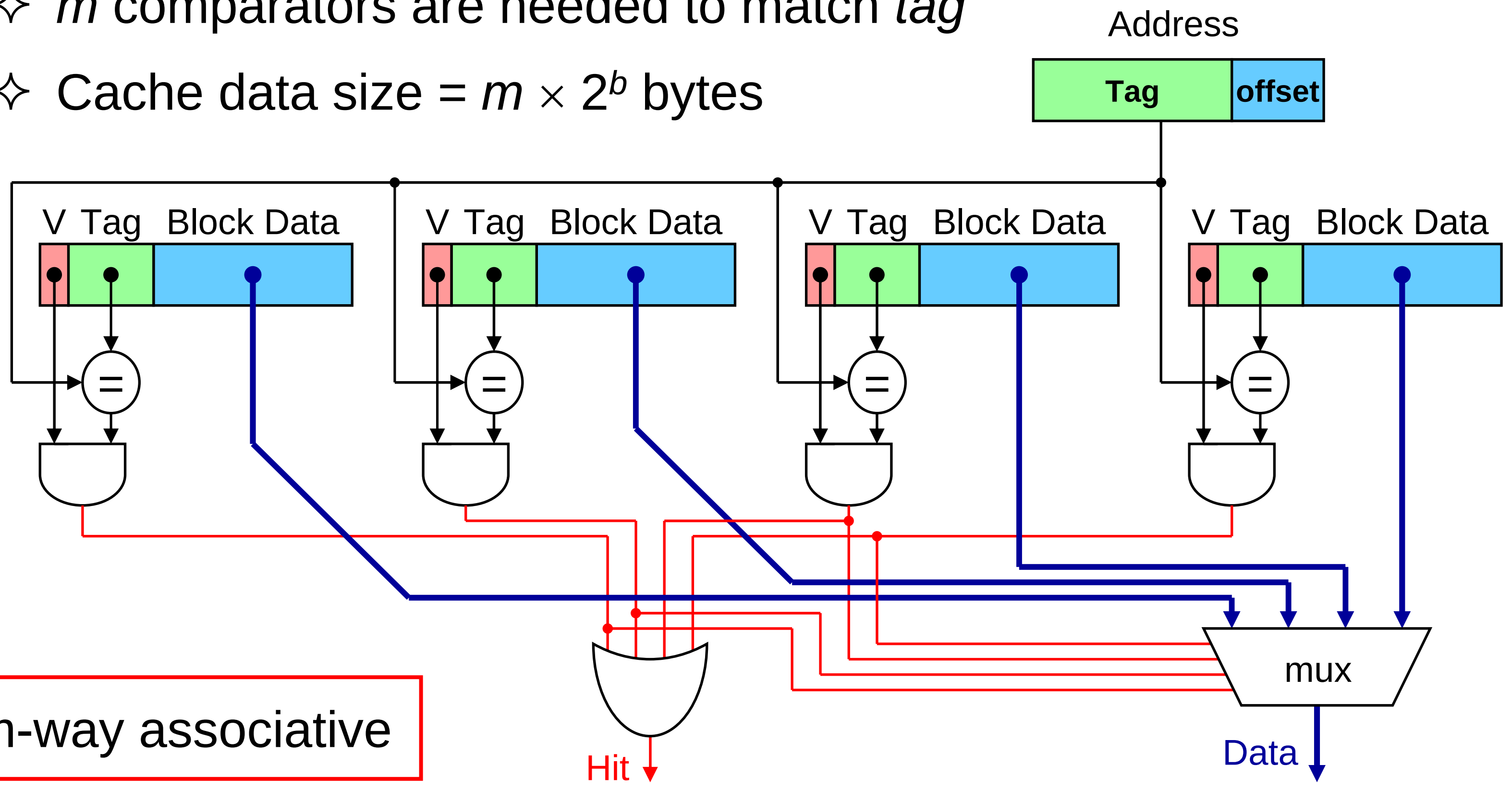
    1000, 1004, 1008, 2548, 2552, 2556.
  - ✧ Map addresses to cache blocks and indicate whether hit or miss
- ❖ Solution:



✧ 1000 = 0x3E8	cache index = 0x1E	Miss (first access)
✧ 1004 = 0x3EC	cache index = 0x1E	Hit
✧ 1008 = 0x3F0	cache index = 0x1F	Miss (first access)
✧ 2548 = 0x9F4	cache index = 0x1F	Miss (different tag)
✧ 2552 = 0x9F8	cache index = 0x1F	Hit
✧ 2556 = 0x9FC	cache index = 0x1F	Hit

Fully Associative Cache

- A block can be placed anywhere in cache ⇒ no indexing
- If *m* blocks exist then
  - *m* comparators are needed to match *tag*
  - Cache data size = $m \times 2^b$ bytes

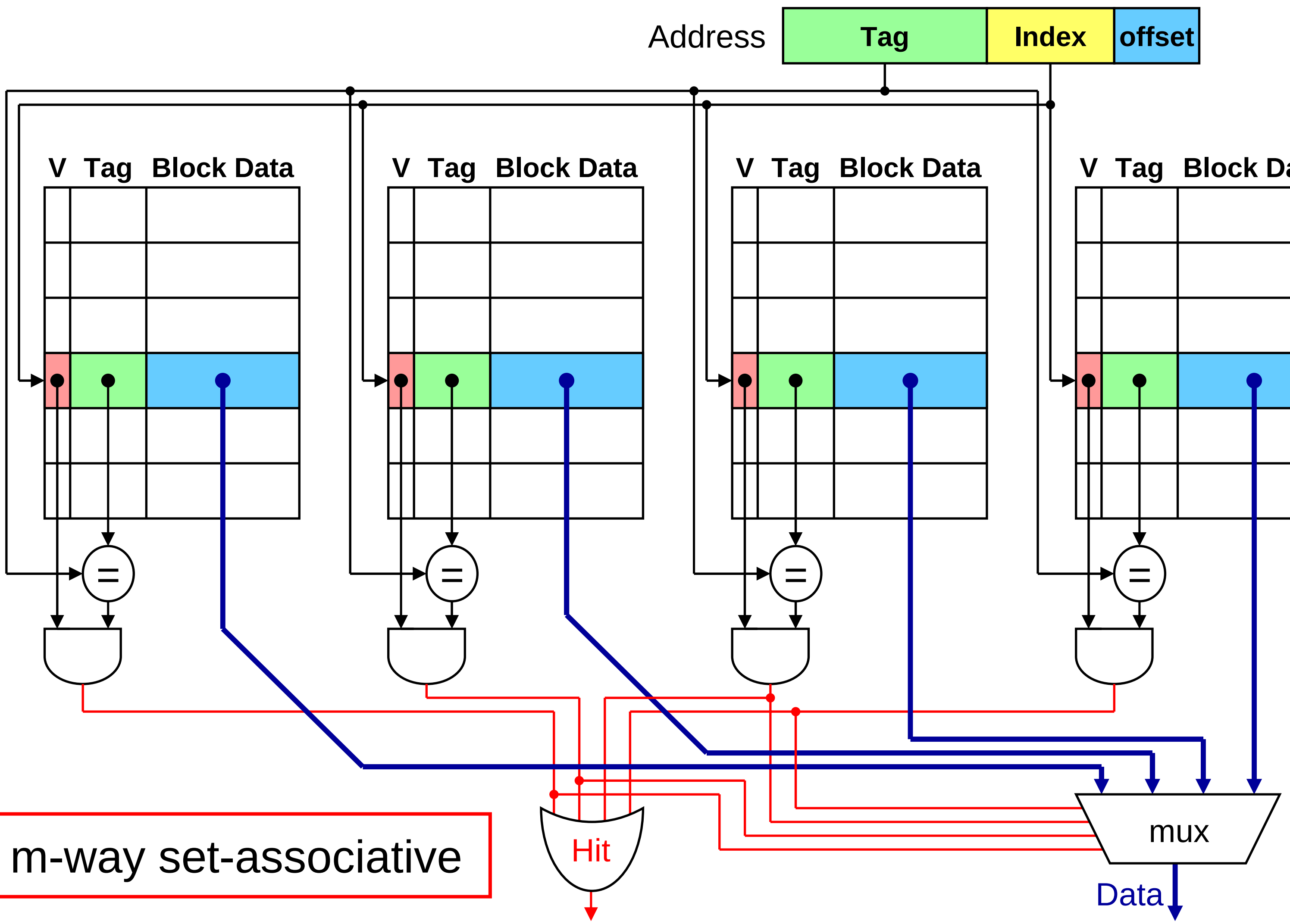


m-way associative

Set-Associative Cache

- A set is a group of blocks that can be indexed
- A block is first mapped onto a set
  - *Set index = Block address* **mod** *Number of sets in cache*
- If there are m blocks in a set (m-way set associative) then
  - m tags are checked in parallel using m comparators
- If 2^n sets exist then set index consists of n bits
- Cache data size = $m \times 2^{n+b}$ bytes (with 2^b bytes per block)
  - Without counting tags and valid bits
- A direct-mapped cache has one block per set ($m = 1$)
- A fully-associative cache has one set ($2^n = 1$ or $n = 0$)

Set-Associative Cache Diagram



m-way set-associative

Write Policy

- Write Through:
  - Writes update cache and lower-level memory
  - Cache control bit: only a Valid bit is needed
  - Memory always has latest data, which simplifies data coherency
  - Can always discard cached data when a block is replaced
- Write Back:
  - Writes update cache only
  - Cache control bits: Valid and Modified bits are required
  - Modified cached data is written back to memory when replaced
  - Multiple writes to a cache block require only one write to memory
  - Uses less memory bandwidth than write-through and less power
  - However, more complex to implement than write through

Write Miss Policy

- What happens on a write miss?
- Write Allocate:
  - Allocate new block in cache
  - Write miss acts like a read miss, block is fetched and updated
- No Write Allocate:
  - Send data to lower-level memory
  - Cache is not modified
- Typically, write back caches use write allocate
  - Hoping subsequent writes will be captured in the cache
- Write-through caches often use no-write allocate
  - Reasoning: writes must still go to lower level memory

Write Buffer

- Decouples the CPU write from the memory bus writing
  - Permits writes to occur without stall cycles until buffer is full
- Write-through: all stores are sent to lower level memory
  - Write buffer eliminates processor stalls on consecutive writes
- Write-back: modified blocks are written when replaced
  - Write buffer is used for evicted blocks that must be written back
- The address and modified data are written in the buffer
  - The write is finished from the CPU perspective
  - CPU continues while the write buffer prepares to write memory
- If buffer is full, CPU stalls until buffer has an empty entry

What Happens on a Cache Miss?

- Cache sends a miss signal to stall the processor
- Decide which cache block to allocate/replace
  - One choice only when the cache is directly mapped
  - Multiple choices for set-associative or fully-associative cache
- Transfer the block from lower level memory to this cache
  - Set the valid bit and the tag field from the upper address bits
- If block to be replaced is modified then write it back
  - Modified block is moved into a Write Buffer
  - Otherwise, block to be replaced can be simply discarded
- Restart the instruction that caused the cache miss
- Miss Penalty: clock cycles to process a cache miss

Replacement Policy

- Which block to be replaced on a cache miss?
- No selection alternatives for direct-mapped caches
- *m* blocks per set to choose from for associative caches
- Random replacement
  - Candidate blocks are randomly selected
  - One counter for all sets (0 to *m* – 1): incremented on every cycle
  - On a cache miss replace block specified by counter
- First In First Out (FIFO) replacement
  - Replace oldest block in set
  - One counter per set (0 to *m* – 1): specifies oldest block to replace
  - Counter is incremented on a cache miss

Replacement Policy - cont'd

- Least Recently Used (LRU)
  - Replace block that has been unused for the longest time
  - Order blocks within a set from least to most recently used
  - Update ordering of blocks on each cache hit
  - With *m* blocks per set, there are *m*! possible permutations
- Pure LRU is too costly to implement when *m* > 2
  - *m* = 2, there are 2 permutations only (a single bit is needed)
  - m = 4, there are 4! = 24 possible permutations
  - LRU approximation are used in practice
- For large *m* > 4,

  Random replacement can be as effective as LRU

Next . . .

- ❖ Random Access Memory and its Structure
- ❖ Memory Hierarchy and the need for Cache Memory
- ❖ The Basics of Caches
- ❖ Cache Performance and Memory Stall Cycles
- ❖ Improving Cache Performance
- ❖ Multilevel Caches

Hit Rate and Miss Rate

- Hit Rate = Hits / (Hits + Misses)
- Miss Rate = Misses / (Hits + Misses)
- I-Cache Miss Rate = Miss rate in the Instruction Cache
- D-Cache Miss Rate = Miss rate in the Data Cache
- Example:
  - Out of 1000 instructions fetched, 150 missed in the I-Cache
  - 25% are load-store instructions, 50 missed in the D-Cache
  - What are the I-cache and D-cache miss rates?
- I-Cache Miss Rate = 150 / 1000 = 15%
- D-Cache Miss Rate = 50 / (25% × 1000) = 50 / 250 = 20%

Memory Stall Cycles

❖ The processor stalls on a Cache miss

- ✧ When fetching instructions from the Instruction Cache (I-cache)
- ✧ When loading or storing data into the Data Cache (D-cache)

Memory stall cycles = Combined Misses × Miss Penalty

❖ Miss Penalty: clock cycles to process a cache miss

Combined Misses = I-Cache Misses + D-Cache Misses

I-Cache Misses = I-Count × I-Cache Miss Rate

D-Cache Misses = LS-Count × D-Cache Miss Rate

LS-Count (Load & Store) = I-Count × LS Frequency

❖ Cache misses are often reported per thousand instructions

Memory Stall Cycles Per Instruction

- Memory Stall Cycles Per Instruction =

  I-Cache Miss Rate × Miss Penalty +

  LS Frequency × D-Cache Miss Rate × Miss Penalty
- Combined Misses Per Instruction =

  I-Cache Miss Rate + LS Frequency × D-Cache Miss Rate
- Therefore, Memory Stall Cycles Per Instruction =

  Combined Misses Per Instruction × Miss Penalty
- Miss Penalty is assumed equal for I-cache & D-cache
- Miss Penalty is assumed equal for Load and Store

Example on Memory Stall Cycles

- Consider a program with the given characteristics
  - Instruction count (I-Count) = 10^6 instructions
  - 30% of instructions are loads and stores
  - D-cache miss rate is 5% and I-cache miss rate is 1%
  - Miss penalty is 100 clock cycles for instruction and data caches
  - Compute combined misses per instruction and memory stall cycles
- Combined misses per instruction in I-Cache and D-Cache
  - $1\% + 30\% \times 5\% = 0.025$ combined misses per instruction
  - Equal to 25 misses per 1000 instructions
- Memory stall cycles
  - 0.025×100 (miss penalty) = 2.5 stall cycles per instruction
  - Total memory stall cycles = $10^6 \times 2.5 = 2{,}500{,}000$

CPU Time with Memory Stall Cycles

$$\text{CPU Time} = \text{I-Count} \times \text{CPI}_{\text{MemoryStalls}} \times \text{Clock Cycle}$$

$$\text{CPI}_{\text{MemoryStalls}} = \text{CPI}_{\text{PerfectCache}} + \text{Mem Stalls per Instruction}$$

- $\text{CPI}_{\text{PerfectCache}}$ = CPI for ideal cache (no cache misses)
- $\text{CPI}_{\text{MemoryStalls}}$ = CPI in the presence of memory stalls
- Memory stall cycles increase the CPI

Example on CPI with Memory Stalls

- A processor has CPI of 1.5 without any memory stalls
  - Average cache miss rate is 2% for instruction and data
  - 50% of instructions are loads and stores
  - Cache miss penalty is 100 clock cycles for I-cache and D-cache
- What is the impact on the CPI?
- Answer:

$$\text{Mem Stalls per Instruction} = \overbrace{0.02 \times 100}^{\textbf{Instruction}} + \overbrace{0.5 \times 0.02 \times 100}^{\textbf{data}} = 3$$

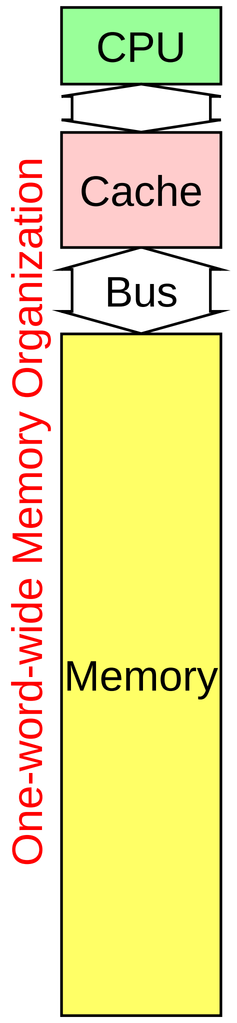
$$\text{CPI}_{\text{MemoryStalls}} = 1.5 + 3 = 4.5 \text{ cycles per instruction}$$

$$\text{CPI}_{\text{MemoryStalls}} / \text{CPI}_{\text{PerfectCache}} = 4.5 / 1.5 = 3$$

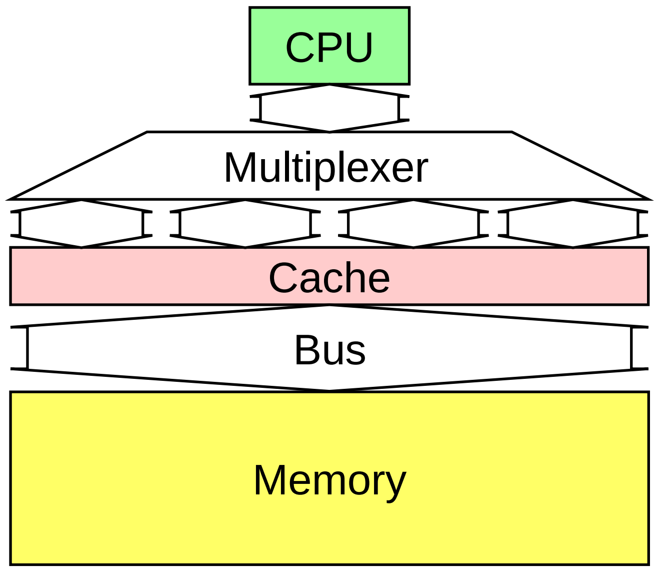
Processor is 3 times slower due to memory stall cycles

$$\text{CPI}_{\text{NoCache}} = 1.5 + (1 + 0.5) \times 100 = 151.5 \text{ (a lot worse)}$$

Designing Memory to Support Caches



One-word-wide Memory Organization

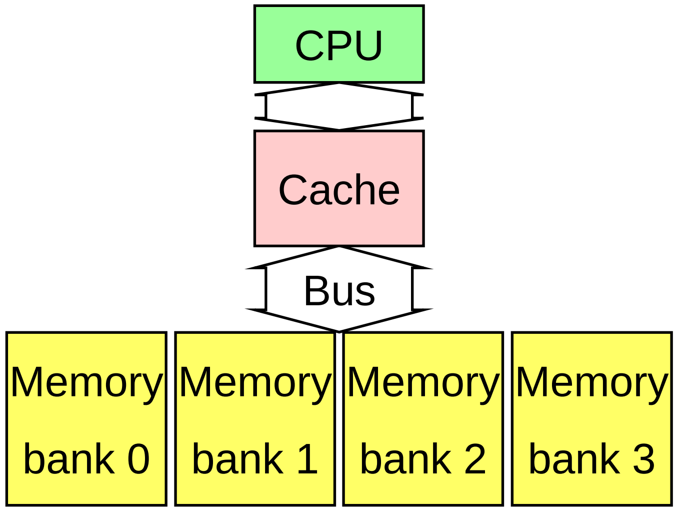


Wide Memory Organization

Wide:
CPU, Mux: 1 word
Cache, Bus, Memory: *N* words
Alpha: 256 bits
Ultra SPARC: 512 bits

One Word Wide:
CPU, Cache, Bus, and Memory have word width: 32 or 64 bits

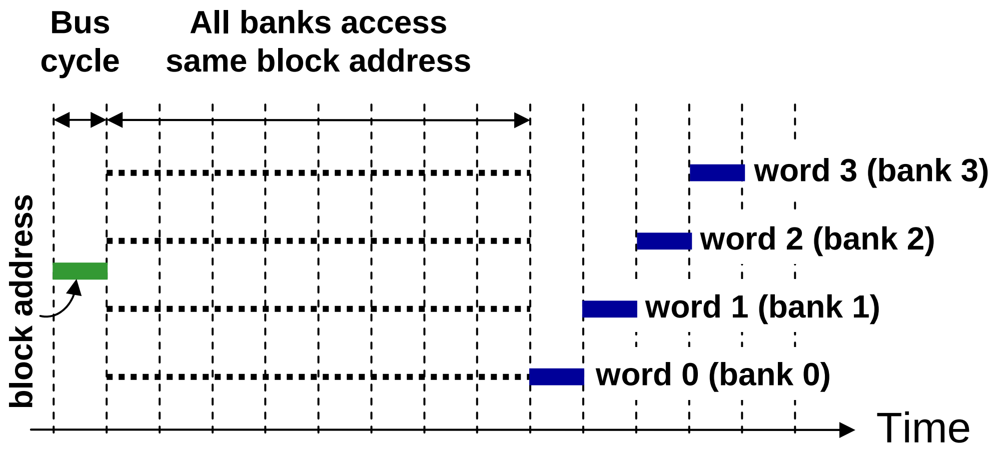
Interleaved:
CPU, Cache, Bus: 1 word
Memory: N independent banks

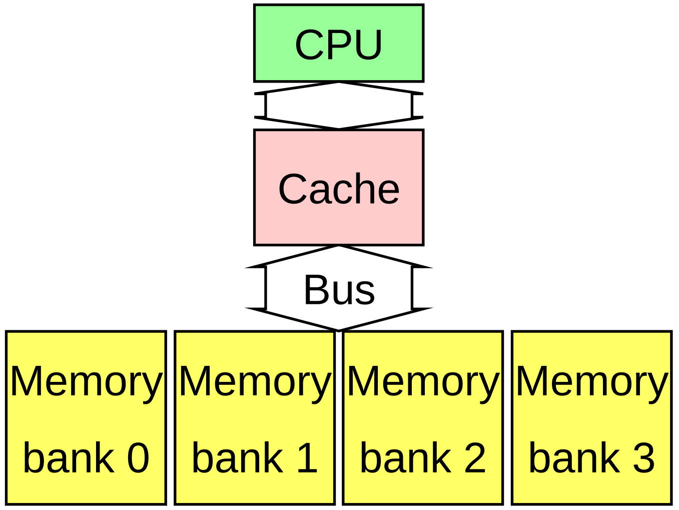


Interleaved Memory Organization

Memory Interleaving

- ❖ Memory interleaving is more flexible than wide access
  - ✧ A block address is sent only once to all memory banks
  - ✧ Words of a block are distributed (interleaved) across all banks
  - ✧ Banks are accessed in parallel
  - ✧ Words are transferred one at a time on each bus cycle





Interleaved Memory Organization

Estimating the Miss Penalty

- Timing Model: Assume the following …
  - 1 memory bus cycle to send address
  - 15 memory bus cycles for DRAM access time
  - 1 memory bus cycle to send data
  - Cache Block is 4 words
- One-Word-Wide Memory Organization

  Miss Penalty = $1 + 4 \times 15 + 4 \times 1 = 65$ memory bus cycles
- Wide Memory Organization (2-word wide)

  Miss Penalty = $1 + 2 \times 15 + 2 \times 1 = 33$ memory bus cycles
- Interleaved Memory Organization (4 banks)

  Miss Penalty = $1 + 1 \times 15 + 4 \times 1 = 20$ memory bus cycles

Next . . .

- ❖ Random Access Memory and its Structure
- ❖ Memory Hierarchy and the need for Cache Memory
- ❖ The Basics of Caches
- ❖ Cache Performance and Memory Stall Cycles
- ❖ Improving Cache Performance
- ❖ Multilevel Caches

Improving Cache Performance

- Average Memory Access Time (AMAT)

  AMAT = Hit time + Miss rate * Miss penalty

- Used as a framework for optimizations
- Reduce the Hit time
  - Small and simple caches
- Reduce the Miss Rate
  - Larger cache size, higher associativity, and larger block size
- Reduce the Miss Penalty
  - Multilevel caches

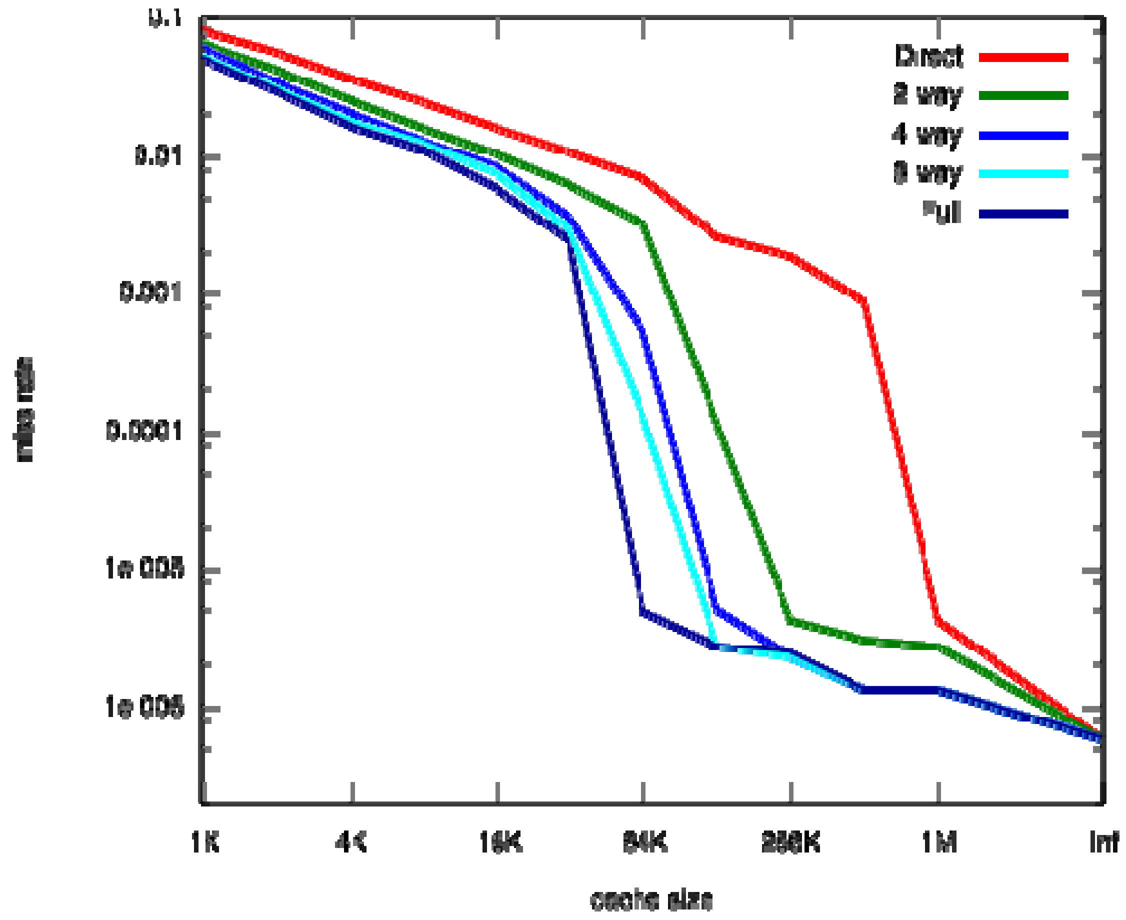
Small and Simple Caches

- Hit time is critical: affects the processor clock rate
  - Fast clock cycle demands small and simple L1 cache designs
- Small cache reduces the indexing time and hit time
  - Indexing a cache represents a time consuming portion
  - Tag comparison also adds to this hit time
- Direct-mapped overlaps tag check with data transfer
  - Associative cache uses additional mux and increases hit time
- Size of L1 caches has not increased much
  - L1 caches are the same size on Alpha 21264 and 21364
  - Same also on UltraSparc II and III, AMD K6 and Athlon
  - Reduced from 16 KB in Pentium III to 8 KB in Pentium 4

Larger Size and Higher Associativity

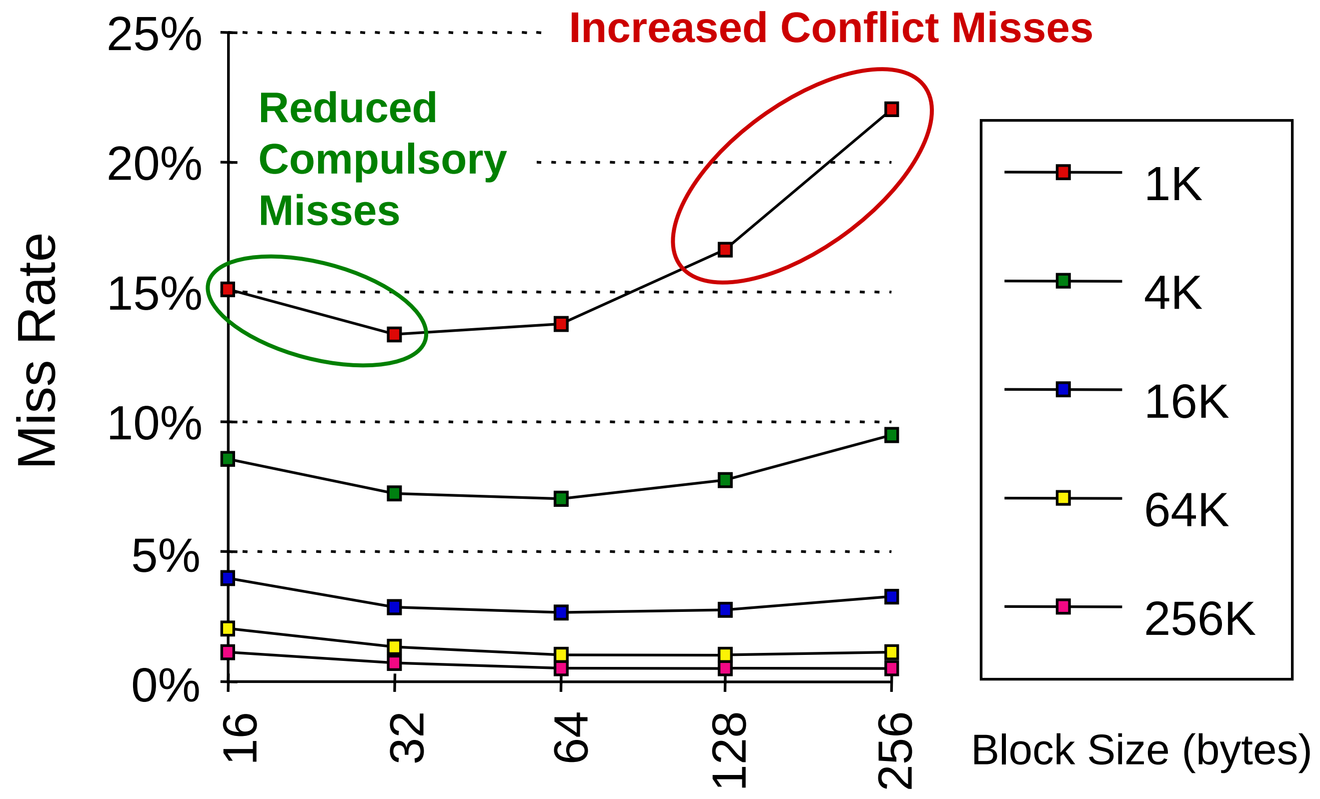
- Cache misses:
  - *Compulsory misses are those misses caused by the first reference to a datum*
  - *Capacity misses* are those misses that occur regardless of associativity or block size, solely due to the finite size of the cache
  - *Conflict misses* are those misses that could have been avoided, had the cache not evicted an entry earlier.
- Increasing cache size reduces capacity misses and conflict misses
- Larger cache size spreads out references to more blocks
- Drawbacks: longer hit time and higher cost
- Larger caches are especially popular as 2nd level caches
- Higher associativity also improves miss rates
  - Eight-way set associative is as effective as a fully associative

Miss rate versus cache size on the Integer portion of SPEC CPU2000



Larger Block Size

- ❖ Simplest way to reduce miss rate is to increase block size
- ❖ However, it increases conflict misses if cache is small



64-byte blocks are common in L1 caches

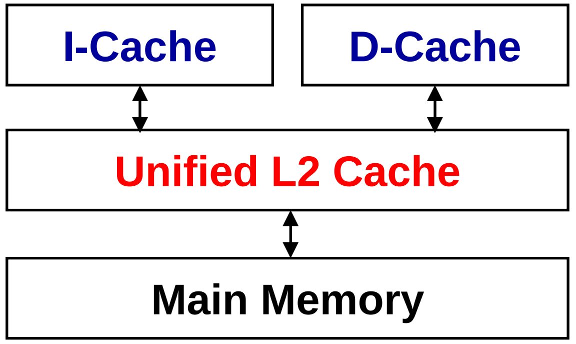
128-byte block are common in L2 caches

Next . . .

- ❖ Random Access Memory and its Structure
- ❖ Memory Hierarchy and the need for Cache Memory
- ❖ The Basics of Caches
- ❖ Cache Performance and Memory Stall Cycles
- ❖ Improving Cache Performance
- ❖ Multilevel Caches

Multilevel Caches

- Top level cache should be kept small to
  - Keep pace with processor speed
- Adding another cache level
  - Can reduce the memory gap
  - Can reduce memory bus loading



- Local miss rate
  - Number of misses in a cache / Memory accesses to this cache
  - Miss Rate_{L1} for L1 cache, and Miss Rate_{L2} for L2 cache
- Global miss rate

  Number of misses in a cache / Memory accesses generated by CPU

  Miss Rate_{L1} for L1 cache, and $\text{Miss Rate}_{L1} \times \text{Miss Rate}_{L2}$ for L2 cache

Multilevel Cache Policies

- Multilevel Inclusion
  - L1 cache data is always present in L2 cache
  - A miss in L1, but a hit in L2 copies block from L2 to L1
  - A miss in L1 and L2 brings a block into L1 and L2
  - A write in L1 causes data to be written in L1 and L2
  - Typically, write-through policy is used from L1 to L2
  - Typically, write-back policy is used from L2 to main memory
    - To reduce traffic on the memory bus
  - A replacement or invalidation in L2 must be propagated to L1

Multilevel Cache Policies – cont'd

- Multilevel exclusion
  - L1 data is never found in L2 cache – Prevents wasting space
  - Cache miss in L1, but a hit in L2 results in a swap of blocks
  - Cache miss in both L1 and L2 brings the block into L1 only
  - Block replaced in L1 is moved into L2
  - Example: AMD Athlon
- Same or different block size in L1 and L2 caches
  - Choosing a larger block size in L2 can improve performance
  - However different block sizes complicates implementation
  - Pentium 4 has 64-byte blocks in L1 and 128-byte blocks in L2

Two-Level Cache Performance – 1/2

- Average Memory Access Time:

  $\text{AMAT} = \text{Hit Time}_{L1} + \text{Miss Rate}_{L1} \times \text{Miss Penalty}_{L1}$

- Miss Penalty for L1 cache in the presence of L2 cache

  $\text{Miss Penalty}_{L1} = \text{Hit Time}_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2}$

- Average Memory Access Time with a 2nd Level cache:

  $\text{AMAT} = \text{Hit Time}_{L1} + \text{Miss Rate}_{L1} \times$

  $(\text{Hit Time}_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2})$

- Memory Stall Cycles per Instruction =

  $\text{Memory Access per Instruction} \times (\text{AMAT} – \text{Hit Time}_{L1})$

Two-Level Cache Performance – 2/2

- Average memory stall cycles per instruction =

  Memory Access per Instruction × Miss Rate_{L1} ×

  (Hit Time_{L2} + Miss Rate_{L2} × Miss Penalty_{L2})
- Average memory stall cycles per instruction =

  Misses per instruction_{L1} × Hit Time_{L2} +

  Misses per instruction_{L2} × Miss Penalty_{L2}
- Misses per instruction_{L1} =

  MEM access per instruction × Miss Rate_{L1}
- Misses per instruction_{L2} =

  MEM access per instruction × Miss Rate_{L1} × Miss Rate_{L2}

Example on Two-Level Caches

- Problem:
  - Miss Rate$_{L1}$ = 4%, Miss Rate$_{L2}$ = 25%
  - Hit time of L1 cache is 1 cycle and of L2 cache is 10 cycles
  - Miss penalty from L2 cache to memory is 100 cycles
  - Memory access per instruction = 1.25 (25% data accesses)
  - Compute AMAT and memory stall cycles per instruction
- Solution:

  AMAT = 1 + 4% × (10 + 25% × 100) = 2.4 cycles

  Misses per instruction in L1 = 4% × 1.25 = 5%

  Misses per instruction in L2 = 4% × 25% × 1.25 = 1.25%

  Memory stall cycles per instruction = 5% × 10 + 1.25% × 100 = 1.75

  Can be also obtained as: (2.4 – 1) × 1.25 = 1.75 cycles