CHAPTER 7-2

Memory

By Pattama Longani
Collage of arts, media and Technology

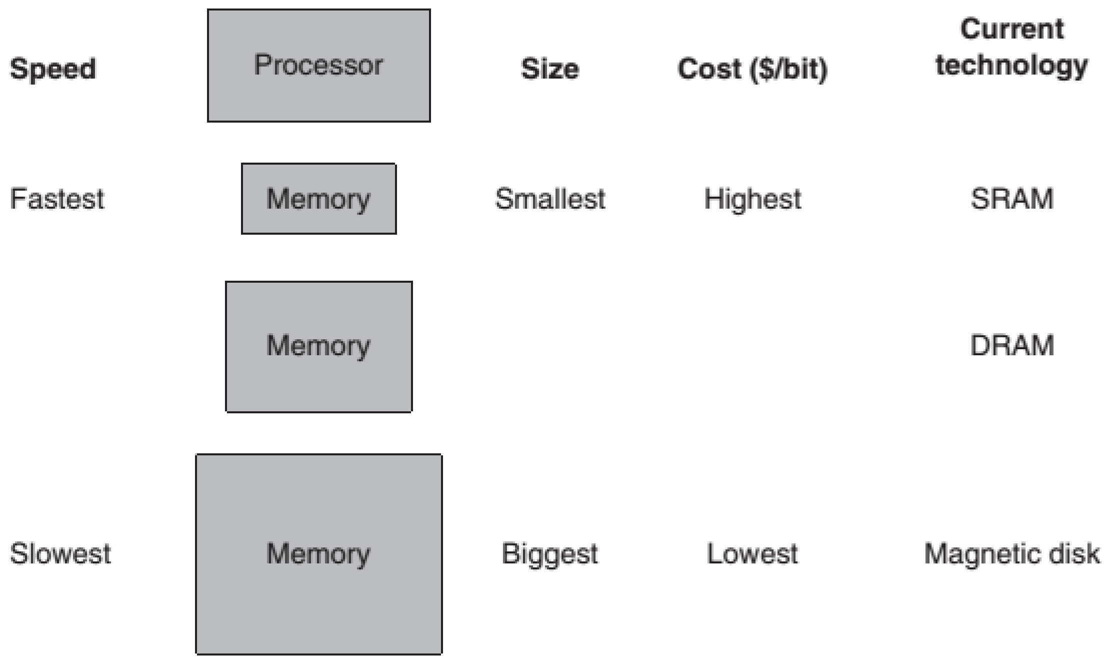
Books in a library

http://www.gettyimages.com/detail/photo/college-students-studying-at-table-royalty-free-image/485207369
http://commons.wikimedia.org/wiki/File:Bookshelf.jpg

memory hierarchy

- A memory hierarchy consists of multiple levels of memory with different speeds and sizes.
- The faster memories are more expensive per bit than the slower memories and thus are smaller.
- *performance* is the major reason for having a memory hierarchy, the time to service hits and misses is important.

memory hierarchy



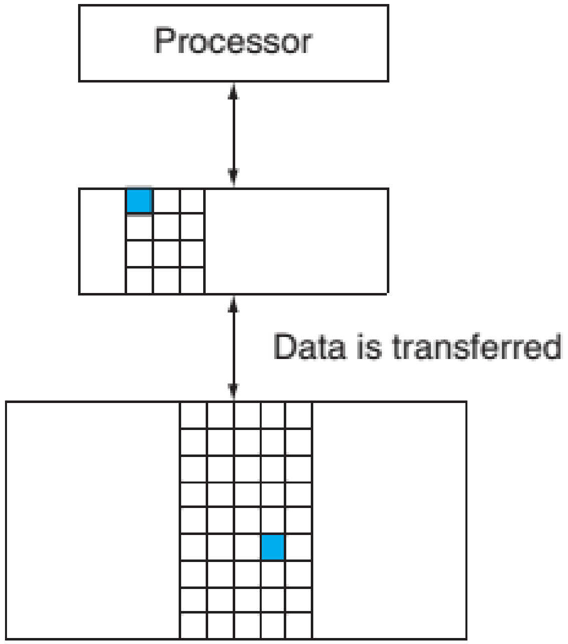
- data is copied between only two adjacent levels at a time

memory hierarchy

- **block** (or line) The minimum unit of information
- **hit rate** The fraction of memory accesses *found* in a level of the memory hierarchy.
- **miss rate** The fraction of memory accesses *not found* in a level of the memory hierarchy.

- **Hit time** is the time to access the upper level of the memory hierarchy, which includes the time needed to determine whether the access is a hit or a miss
- **miss penalty** is the time to replace a block in the upper level with the corresponding block from the lower level, plus the time to deliver this block to the processor

memory hierarchy



- Because the upper level is smaller and built using faster memory parts, the hit time will be much smaller than the time to access the next level in the hierarchy,

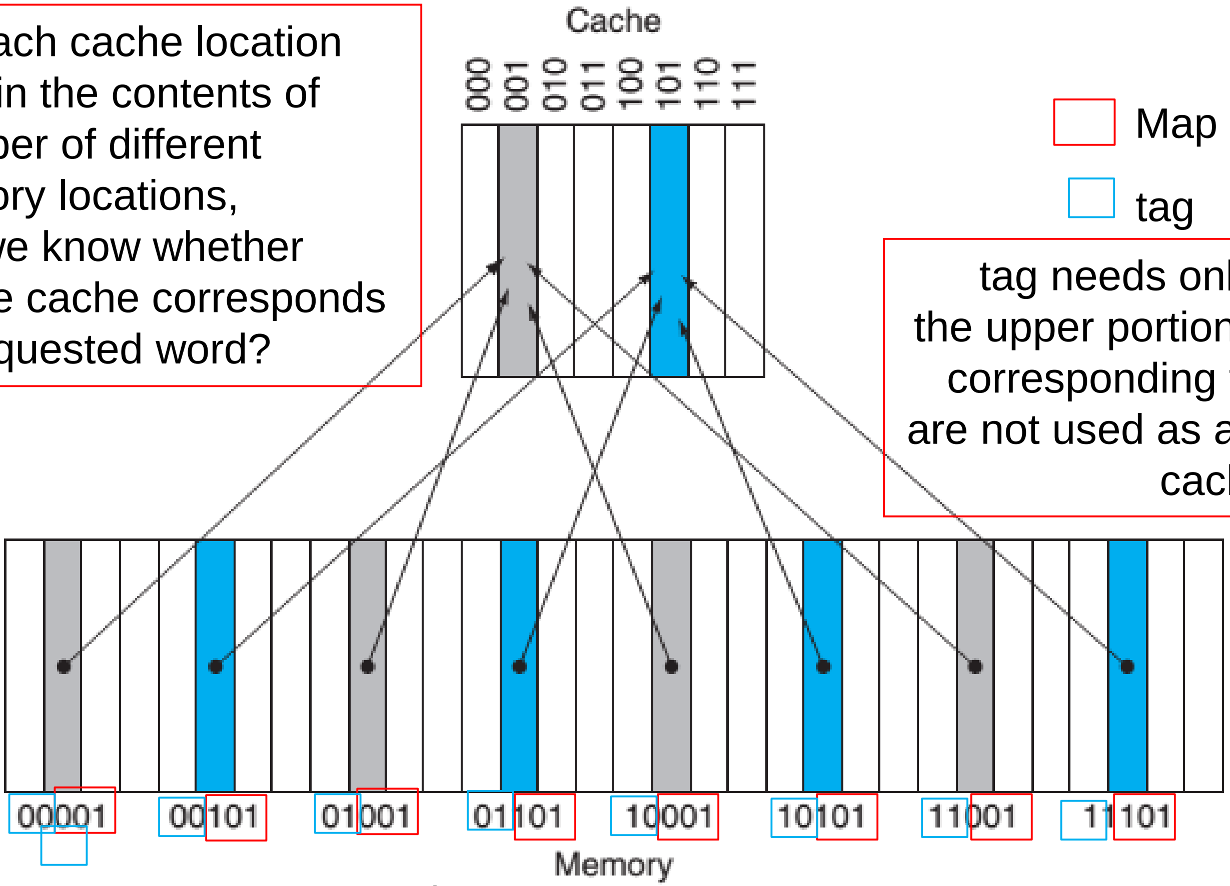
memory systems affection

- how the operating system manages memory and I/O
- how compilers generate code
- how applications use the computer

Cache

(Block address) modulo (Number of blocks in the cache)

Because each cache location can contain the contents of a number of different memory locations, how do we know whether the data in the cache corresponds to a requested word?



🟥 Map to cache

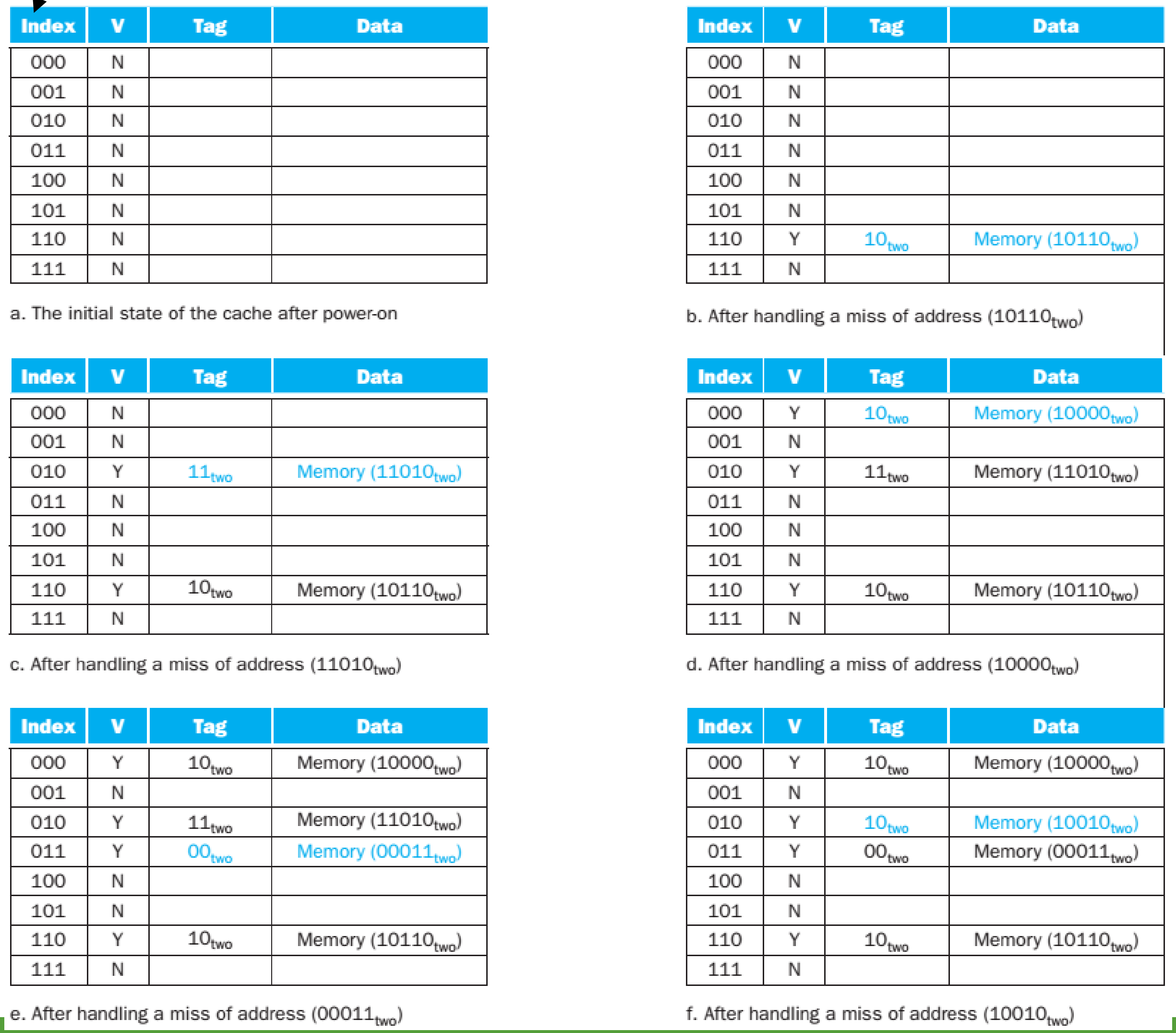
🟦 tag

tag needs only to contain the upper portion of the address, corresponding to the bits that are not used as an index into the cache.

Check memo addr

Accessing a Cache

Index field used is used as an address to reference the cache



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

a. The initial state of the cache after power-on

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

b. After handling a miss of address (10110_{two})

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

c. After handling a miss of address (11010_{two})

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

d. After handling a miss of address (10000_{two})

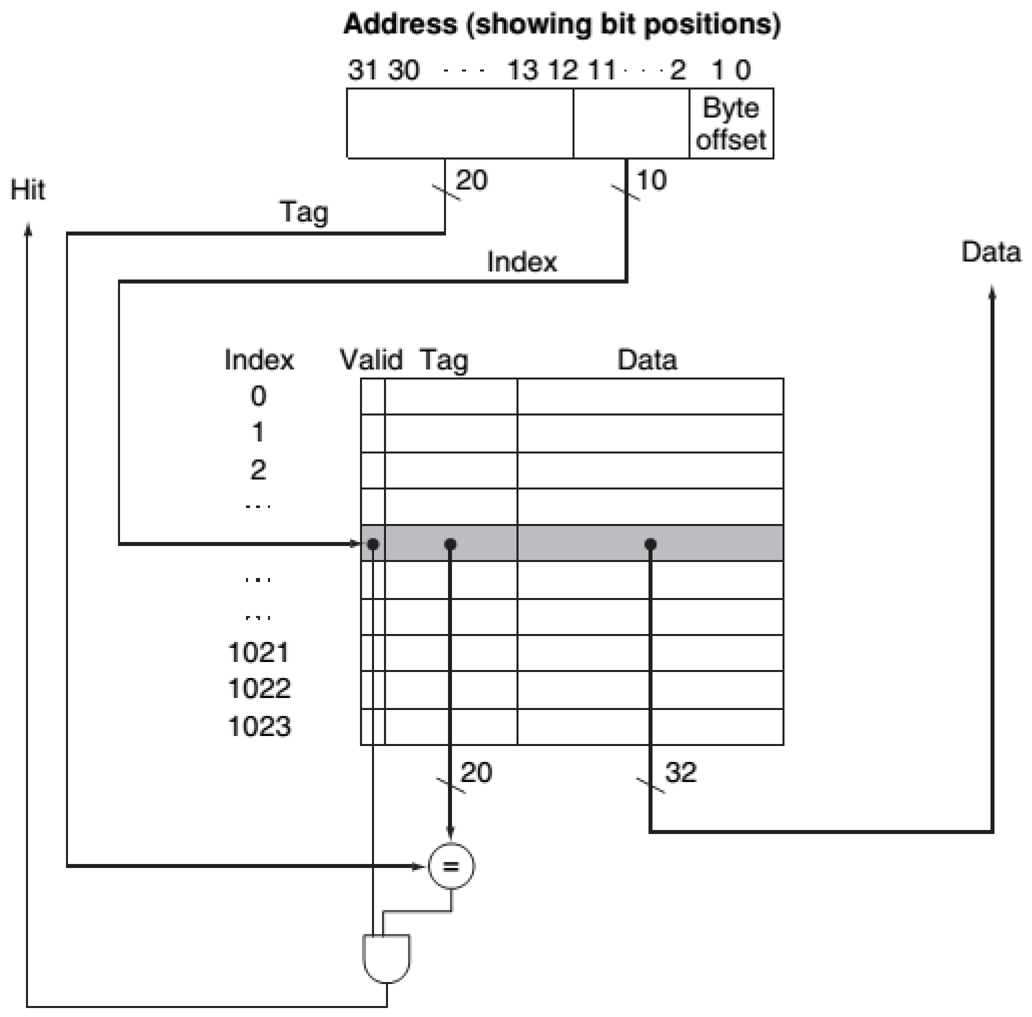
Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

e. After handling a miss of address (00011_{two})

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

f. After handling a miss of address (10010_{two})

Accessing a Cache



MIPS cache

- Since words are aligned to multiples of four bytes, the least significant two bits of every address specify a byte within a word.
- Hence, the **least significant two bits are ignored** when selecting a word in the block.
- The total number of bits needed for a cache is a function of the **cache size** and the **address size**.
- Cache includes both the storage for the **data** and the **tags**.

MIPS cache

- 32-bit byte addresses
- A direct-mapped cache
- The cache size is 2^n blocks, so n bits are used for the index
- The block size is 2^m words (2^{m+2} bytes), so m bits are used for the word within the block, and two bits are used for the byte part of the address

the size of the tag field is

Byte offset

$$32 - (n + m + 2).$$

The total number of bits in a direct-mapped cache is

$$2^n \times (\text{block size} + \text{tag size} + \text{valid field size}).$$

Ex : Bits in a Cache

How many total bits are required for a direct-mapped cache with 16 KB of data and 4-word blocks, assuming a 32-bit address?

- 16 KB = 4K (2^{12}) words.

Different data

- With a block size of 4 words (2^2), there are 1024 (2^{10}) blocks. -> cache size n = 10
- Each block has 4 *32 or 128 bits of data
- Tag = 32 – 10 – 2 – 2 bits
- valid bit = 1 bit.
- Thus, the total cache size is

$2^{10} * (4 *32 + (32 - 10 - 2 - 2) + 1) = 2^{10} *147 = 147$ Kbits

Handling Cache Misses

- The cache miss handling is done in collaboration with the processor control unit and with a separate controller that initiates the memory access and refills the cache.
- The processing of a cache miss creates a pipeline stall.
- As opposed to an **interrupt**, which would require saving the state of all registers.

Handling Cache Misses

- For a cache miss, we can stall the entire processor, essentially freezing the contents of the temporary and programmer-visible registers, while we wait for memory.
- More sophisticated out-of-order processors can allow execution of instructions while waiting for a cache miss, but we'll assume in-order processors that stall on cache misses in this section.

Handling Cache Misses

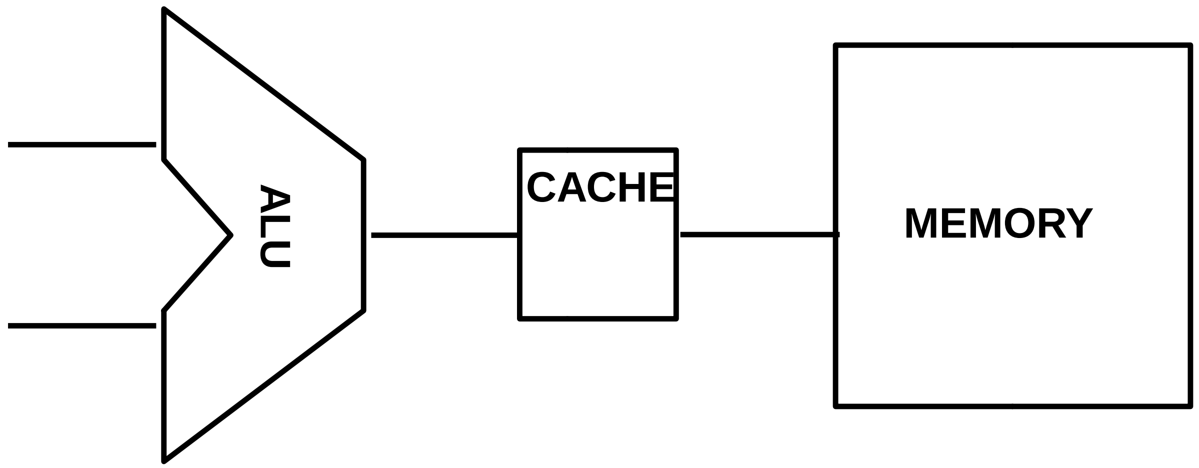
Steps to be taken on an instruction cache miss:

- 1. Send the original PC value (PC= PC–4) to the memory.
- 2. Instruct main memory to perform a read and wait for the memory to complete its access.

Handling Cache Misses

- 3. Write the cache entry,
  - putting the data from memory in the data portion
  - writing the upper bits of the address into the tag field
  - turning the valid bit on
- 4. Restart the instruction execution at the first step
  - refetch the instruction, this time finding it in the cache.

Handling Writes



- **write-through** A scheme in which writes always update both the cache and the next lower level of the memory hierarchy, ensuring that data is always consistent between the two.

Handling Writes

- Write-through (every write causes the data to be written to main memory) can takes a long time.
- **For example**
  - suppose 10% of the instructions are stores. If the CPI without cache misses was 1.0, spending 100 extra cycles on every write would lead to a CPI of 1.0 + 100 *10% = 11
  - Reducing performance by more than a factor of 10.

Handling Writes

- One solution to this problem is to use a **write buffer.**
- *A write buffer stores the data while it is waiting to be written to memory.*
  - After writing the data into the cache and into the write buffer, the processor can continue execution.
  - When a write to main memory completes, the entry in the write buffer is freed.
  - If the write buffer is full when the processor reaches a write, the processor must stall until there is an empty position in the buffer

Handling Writes

- if the rate at which the memory can complete writes is less than the rate at which the processor is generating writes, no amount
- Stalls may still occur if the rate at which writes are generated may be *less* than the rate at which the memory can accept them.
- To reduce the occurrence of such stalls, processors usually increase the depth of the write buffer beyond a single entry.

Handling Writes

- Another alternative to a write-through scheme is a scheme called **write-back**
    - 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.
- Hardware modifier
- more complex

Measuring & Improving Cache Performance

- reducing the miss rate by reducing the probability that two different memory blocks will contend for the same cache location.
- reduces the miss penalty by adding an additional level to the hierarchy, *multilevel caching* technique

Measuring & Improving Cache Performance

- CPU time can be divided into the clock cycles that the CPU spends executing the program and the clock cycles that the CPU spends waiting for the memory system.

$$\text{CPU time} = (\text{CPU execution clock cycles} + \text{Memory-stall clock cycles}) \times \text{Clock cycle time}$$

Measuring & Improving Cache Performance

- We make an assumption that the memory-stall clock cycles come primarily from cache misses.
- Memory-stall clock cycles can be defined as the sum of the stall cycles coming from reads plus those coming from writes:

$$\text{Memory-stall clock cycles} = \text{Read-stall cycles} + \text{Write-stall cycles}$$

Measuring & Improving Cache Performance

$$\text{Read-stall cycles} = \frac{\text{Reads}}{\text{Program}} \times \text{Read miss rate} \times \text{Read miss penalty}$$

$$\text{Write-stall cycles} = \left(\frac{\text{Writes}}{\text{Program}} \times \text{Write miss rate} \times \text{Write miss penalty}\right) + \text{Write buffer stalls}$$

Measuring & Improving Cache Performance

- Because the write buffer stalls depend on the proximity of writes, and not just the frequency, it is not possible to give a simple equation to compute such stalls.
- the write buffer stalls will be small, and we can safely **ignore them**.

Measuring & Improving Cache Performance

- In most write-through cache organizations, the read and write miss penalties are the same, we can combine the reads and writes by using a single miss rate and the miss penalty

$$\text{Memory-stall clock cycles} = \frac{\text{Memory accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalty}$$

$$\text{Memory-stall clock cycles} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}$$

Measuring & Improving Cache Performance

- Assume the miss rate of an instruction cache is 2% and the miss rate of the data cache is 4%. If a processor has a CPI of 2 without any memory stalls and the miss penalty is 100 cycles for all misses, determine how much faster a processor would run with a perfect cache that never missed. Assume the frequency of all loads and stores is 36%.

Measuring & Improving Cache Performance

$$\text{Instruction miss cycles} = I \times 2\% \times 100 = 2.00 \times I$$

$$\text{Data miss cycles} = I \times 36\% \times 4\% \times 100 = 1.44 \times I$$

The total number of memory-stall cycles is 2.00 I + 1.44 I = 3.44 I

Total CPI = 2 + 3.44 = 5.44

$$\frac{\text{CPU time with stalls}}{\text{CPU time with perfect cache}} = \frac{I \times \text{CPI}_{\text{stall}} \times \text{Clock cycle}}{I \times \text{CPI}_{\text{perfect}} \times \text{Clock cycle}}$$

$$= \frac{\text{CPI}_{\text{stall}}}{\text{CPI}_{\text{perfect}}} = \frac{5.44}{2}$$

The performance with the perfect cache is better by $\frac{5.44}{2} = 2.72$.

Measuring & Improving Cache Performance

- Memory system is related to many part of computer.

What happens if the processor is made faster, but the memory system is not?

Measuring & Improving Cache Performance

Ex: Suppose we speed-up the computer in the previous example by reducing its CPI from 2 to 1 without changing the clock rate, which might be done with an improved pipeline.

- The system with cache misses would then have a CPI of 1 + 3.44 = 4.44
- Speed up = $\frac{4.44}{1} = 4.44$

Measuring & Improving Cache Performance

The amount of execution time spent on memory stalls would have risen from

$$3.44/5.44 = 63\%$$

to

$$3.44/4.44 = 77\%.$$

Measuring & Improving Cache Performance

- To capture the fact that the time to access data for both hits and misses affects performance, designers sometime use *average memory access time* (AMAT) as a way to examine alternative cache designs.
- Average memory access time is the average time to access memory considering both hits and misses and the frequency of different accesses; it is equal to the following:

$$\text{AMAT} = \text{Time for a hit} + \text{Miss rate} \times \text{Miss penalty}$$

Example

- Find the AMAT for a processor with a 1 ns clock cycle time, a miss penalty of 20 clock cycles, a miss rate of 0.05 misses per instruction, and a cache access time (including hit detection) of 1 clock cycle. Assume that the read and write miss penalties are the same and ignore other write stalls

$$
\begin{aligned}
\text{AMAT} &= \text{Time for a hit} + \text{Miss rate} \times \text{Miss penalty} \\
&= 1 + 0.05 \times 20 \\
&= 2 \text{ clock cycles}
\end{aligned}
$$

or 2 ns.