Writing

- Different write approaches
- Write-through Approach
  - update both cache and main memory for every write
  - each write takes longer
- Write-buffered Approach
  - dedicated buffer that stores the writes to be made to memory
- Write-back approach
  - only update cache block, mark cache block as dirty
  - update main memory when the cache block is replaced
Cache Performance

CPUPTime = (CPU Execution cycles + Mem Stall Cycles) x Cycle Time

Memory stall cycles

= \frac{\text{Memory accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalty}

= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}
Cache Performance Example

What is the CPI?

- instruction cache miss rate = 2%
- data cache miss rate = 4%
- miss penalty = 100 cycles
- base CPI (ideal cache) = 2
- load/store are 36% of instructions

ANS: miss cycles per instruction:

- instruction: $0.02 \times 100 = 2$
- data: $0.36 \times 0.04 \times 100 = 1.44$
- actual CPI = $2 + 2 + 1.44 = 5.44$
Cache Performance Example (Contd..)

So how much faster is a perfect cache?

\[
\frac{\text{CPUTimewithStalls}}{\text{CPUTimePerfect}} = \frac{\text{IC} \times \text{CPIwithStalls} \times \text{CycleTime}}{\text{IC} \times \text{CPIPerfect} \times \text{Cycletime}}
\]

\[
= \frac{\text{CPIwithStalls}}{\text{CPIPerfect}}
\]

\[
= 5.44/2
\]

\[
= 2.72 \text{ times faster!}
\]
Average Access Time

- average memory access time (AMAT)
  - $AMAT = \text{hit time} + \text{miss rate} \times \text{miss penalty}$
- Example
  - 1ns clock time, hit time = 1 cycle, miss penalty = 20 cycles, instruction cache miss rate = 5%
  - $AMAT = 1 \times 1 + 0.05 \times 20 \times 1 = 2\text{ns}$
    - 2 cycles per instruction
Cache Performance Summary

- when CPU performance is increased
  - miss penalty becomes more significant
- cache behaviour
  very important for system performance!
Associative Caches

- fully associative
  - allow a given memory block to go in any cache block
  - No direct mapping from memory block no. to cache block
  - requires all entries to be searched at once
  - hardware comparator per entry (expensive)

- n-way set associative
  - Cache blocks are grouped into sets. Each set has $n$ blocks.
  - A memory block can go into any slot
  - Memory block number determines which set
    - (block number) mod (# sets in cache)
  - only search entries in a given set – cheaper
Associative Caches

Direct mapped

Set associative

Fully associative

Block #

0 1 2 3 4 5 6 7

Data

Set #

0 1 2 3

Data

Tag

1 2

Search

Tag

1 2

Search
Example: Comparison of the three Caching schemes

- A cache with 4 blocks
  - Memory block access sequence: 0, 8, 0, 6, 8
  - Direct mapped:

<table>
<thead>
<tr>
<th>Memory Block Address</th>
<th>Cache Index</th>
<th>Hit/Miss</th>
<th>Cache Content After Access</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>miss</td>
<td>Mem[0]</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>miss</td>
<td>Mem[8]</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>miss</td>
<td>Mem[0]</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>miss</td>
<td>Mem[0] Mem[6]</td>
</tr>
</tbody>
</table>
Example: Comparison of the three Caching schemes

- 2-way set associative
  - Each set has 2 blocks (it is just coincidence that there are 2 sets)

<table>
<thead>
<tr>
<th>Block Address</th>
<th>Cache Index</th>
<th>Hit/Miss</th>
<th>Cache Content After Access</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Set 0</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Set 1</strong></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>miss</td>
<td>Mem[0]</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>miss</td>
<td>Mem[0] Mem[8]</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>hit</td>
<td>Mem[0] Mem[8]</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>miss</td>
<td>Mem[0] Mem[6]</td>
</tr>
</tbody>
</table>
### Associativity Example

- **fully associative**

<table>
<thead>
<tr>
<th>Block Address</th>
<th>Hit/Miss</th>
<th>Cache Content After Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>miss</td>
<td>Mem[0]</td>
</tr>
<tr>
<td>8</td>
<td>miss</td>
<td>Mem[0] Mem[8]</td>
</tr>
<tr>
<td>0</td>
<td>hit</td>
<td>Mem[0] Mem[8]</td>
</tr>
</tbody>
</table>
How Much Associativity

- Increased associativity decreases miss rate
  - but with diminishing return (Why?)
- Higher Cost, Smaller Cache
- Higher Associativity may lead to higher hit time
Replacement Policy: Which block to replace from the set?

- Direct mapped: no choice
- Associative: evict *least recently used* (LRU)
  - difficult/costly with increasing associativity
- Alternative: random replacement
  - simple and fast, not too much worse than LRU
- Combination LRU+Random
Multilevel Caches

- primary (level-1) cache attached to CPU
  - small, but fast (usually instruction speed)
- level-2 cache services misses from L1 cache
  - larger, slower, but still faster than main memory
- main memory services L2 cache misses
- some high-end systems have even more levels
- on-chip vs. off-chip placement of cache
- If L2 contains the requested data item, Miss penalty for L1 will be the access time for L2. The Miss Penalty will be higher if data not in L2 either.
Multilevel Cache Example

- assume
  - CPU base CPI = 1, clock rate = 4GHz
  - miss rate = 2%
  - main memory access time = 100ns
- with just primary cache
  - miss penalty = 100ns/0.25ns = 400 cycles
  - effective CPI = 1 + 0.02 * 400 = 9
Multilevel Cache Example

- now add L2 cache
  - access time = 5ns
  - global miss rate to main memory = 0.5%
- primary miss with L2 hit
  - penalty = 5ns/0.25ns = 20 cycles
- primary miss with L2 miss
  - extra penalty = 400 cycles

\[ \text{CPI} = 1 + 0.02 \times 20 + 0.005 \times 400 = 3.4 \]

Processor with L2 cache is \( 9/3.4 = 2.6 \) times faster!
I/O Devices

- I/O devices can be characterized by
  - behaviour: input, output, storage
  - partner: human or machine
  - data rate: bytes/sec, transfers/sec

- I/O device vs. I/O controller
  - controller: processor that controls device
I/O System Characteristics

- dependability important – especially for storage
- performance
  - latency (response time)
  - throughput (bandwidth)
- desktops & embedded systems
  - mainly interested in response time & device diversity
- servers
  - mainly interested in throughput & expandability
Interconnecting Components

- need interconnection between
  - CPU, memory, I/O controllers
- Bus: shared communication channel
  - parallel set of wires for data and synchronization
- performance limited by physical factors
  - wire length, number of connections
I/O Bus

- Processor
- Cache
- Memory-I/O Interconnect
  - Main memory
  - I/O controller
    - Disk
  - I/O controller
    - Graphics output
  - I/O controller
    - Network

Interrupts
Bus Types

- processor-memory bus
  - Connects processor to main-memory
  - short, high-speed
  - design matches memory organization
- I/O bus: connect different devices together
  - longer, multiple connections
  - interoperability standards
- I/O Bus does not directly connect to Memory. **Bridge** connects memory & I/O bus
Bus Signals

- data lines
  - carry addresses and data
  - Multiplexed (shared) or separate
- control lines: carries info about the data being carried
  - indicate data types, synchronize transactions
- Serial Communication: One bit at a time on one line
- Parallel Comm: Multiple bits at a time, sent on many parallel lines (still one bit on one line at a time)
Bus Synchronization

• The information being sent back and forth on a bus needs to be synchronized so that the component making a request knows when it is receiving the response.

• **Synchronous buses** include a clock in the control lines, and follow a fixed protocol for communicating.
  
  • For example: Memory will receive the address and read on the first clock cycle, then memory is required to respond with the data word in question on the sixth cycle.

• Every device on the bus must run at the same clock rate

• Clock skew is an issue, limits our buses length

• **Asynchronous Buses** are not clocked, instead agree upon some kind of a handshaking protocol: *Send out signal, wait for acknowledgement before sending next*

• Can support any device that follows protocol, doesn't need clocks, Can be lengthened as needed