Taking advantage of spatial locality

Use block size larger than one word
Example: two words

Alternate representations

\[
\text{Word Index} = (\text{memory word address}) \mod (\text{cache size in words})
\]

\[
\text{Block Index} = (\text{memory block address}) \mod (\text{cache size in blocks})
\]

- One tag per block \(\implies\) improves efficiency
- write misses have to be handled carefully (should preserve block integrity)
  - copy block from memory to cache
  - write the word within the block in cache.

Example: block size = 2 words

When a CPU issues a memory address, it is decomposed into a tag, a block index and a block offset.
Taking advantage of spatial locality (block size = 4)

- What is the total number of bits needed for a 64KB cache assuming 32-bit address?
- What about an 8KB cache?

Hardware Issues

- Make reading multiple words faster by using banks of memory

EX: 1 cycle to send a word or an address and 15 cycles to access a word in memory—what is the miss penalty? (block = 4 words).

<table>
<thead>
<tr>
<th>Organization</th>
<th>Time to Access Memory</th>
<th>Time to Access Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. one-word wide memory</td>
<td>$4 \times (1 + 15 + 1) = 68$ cycles</td>
<td>$1 + 15 + 1 = 17$ cycles</td>
</tr>
<tr>
<td>b. 4-word wide organization</td>
<td>$1 + 15 + 1 = 17$ cycles</td>
<td>$(1 + 15 + 1) \div 3 = 20$ cycles</td>
</tr>
<tr>
<td>c. Interleaved memory</td>
<td>$1 + 15 + 1 = 17$ cycles</td>
<td>$(1 + 15 + 1) \div 3 = 20$ cycles</td>
</tr>
</tbody>
</table>
### Interleaved memory

- $T = 1$ send request to bank 0
- $T = 2$ send request to bank 1
- $T = 3$ send request to bank 2
- $T = 4$ send request to bank 3
- $T = 16$ data ready at Bank 0 and received at $T = 17$
- $T = 17$ data ready at Bank 0 and received at $T = 18$
- $T = 18$ data ready at Bank 0 and received at $T = 19$
- $T = 19$ data ready at Bank 0 and received at $T = 20$

### CPU control with caches

- In the CPU architecture of Chapters 4, replace the memory modules by caches -- for example, in pipelined architecture, it takes one cycles to read/write a word into/from the Instruction cache or the data cache.
- The hit/miss signal from the cache can go to the CPU control.
- On a read miss, the pipeline is stalled and the cache controller
  - sends the address and a read command to the main memory,
  - may have to make room for the new block
  - puts data in cache, and return the missed word to the CPU
  - continues execution.
- On a write hit, the cache controller
  - writes the word into the cache, and
  - If a WRITE THROUGH cache, writes the word also in main memory.
CPU control with caches

- On a write miss, the cache controller
  - If a WRITE BACK cache
    - Brings the block to cache (by sending the address and a read command to memory) – needed if a block is larger than one word.
    - Writes the word into the cache
  - If a WRITE THROUGH cache, writes the word in main memory.
    - If “No Write Allocate”, does not allocate the block in the cache
    - If “Write Allocate”,
      - Brings the block to cache
      - Writes the word into the cache

Handling writes

- The WRITE BACK scheme:
  - Does not write into memory every write hit,
  - Writes the block back into memory when done with it (when is that ??).
  - Writes the block back only if it is dirty (have been written on)
  - For some time, cache and memory are incoherent (any problems??)

- Write buffers are used to avoid stalling the pipeline while a word or a block is written into memory.
  - What if another write or read occurs before memory is done with the first write? -- a typical speed mismatch problem.
  - Do we ever have to stall the execution on a write miss?

What kind of locality have we taken advantage of?
Decreasing miss ratio with associativity

Example (2-way set associative, Block size = 1)

Address = 100 01

Equivalent to having two direct mapped caches, with the possibility of putting the data in either of the two caches

- Need a block replacement policy: LRU (complex) or an access bit (simple)
- Give a series of references for which direct mapped cache results in a lower miss ratio than a 2-way set associative cache (with LRU). What about a higher miss ratio?
An implementation (4KB cache, 4-way associative, block size = 1)

- Associativity implies more hardware cost.
- Compare the overhead (for tags) with that in direct mapping,
- May use block size larger than one word,

An implementation (4KB cache, 2-way associative, block size = 2)

- Equivalent to two caches, each with block size = 2.
- Each cache has 255 blocks. That is word index = 9 bits, 8 for block index and one for offset within a block,