Spatial Locality

- one word lines do not exploit spatial locality for typical computations

- spatial locality implies if address \( A \) is needed then \( A + \delta \) will be needed soon.

- Caches attempt to exploit spatial locality by using multiple word cache lines.

- whenever a miss occurs on address \( A \) then a neighborhood around \( A \) is load into cache.
insert figure 7.10
• 32 bit address

• bits 0 and 1 are byte offset in word (used in processor/ cache communication)

• bits 2 and 3 are word offset in line since there are four words per line. (used by the cache to select which word to return to processor as data)

• bits 4 to 31 are the line or block address. Theses 28 bits uniquely identify each block in the physical address space.

• bits 4 to 15 are the 12 bit block index since there are 4K blocks in the cache.

• bits 16 to 31 are the tag.
• note there is only one tag per block (not one per word)

• note that the lines’ address are always aligned with addresses that have 0 in bits 0 through 3 since there are 16 bytes per line.

• if you access the third word in a line (address $A$) then words $A - 2$, $A - 1$, $A$ and $A + 1$ are fetched on a miss, i.e., $A$ does not indicate the first word in the line fetched.
Example

- Direct-mapped, 4 blocks, 2 words per block.

- address: tag (31 - 5), block index (4-3), block offset (2), word offset (1-0)

- we will work with all word aligned addresses so offset will refer to concatenation of block and word offset (2-0)

- Consider the following sequence of HEX addresses accessing the cache.

- 0x4, 0x8, 0xc, 0x0, 0x20, 0x4, assuming cold start
- 0x4, 0 – 000100

- tag 0x0, index 0, offset 0x4

- miss, fetches words 0x0 and 0x4 into block 0.

- 0x8, 0 – 001000

- tag 0x0, index 01, offset 0x0

- miss, fetches words 0x8 and 0xc into block 1.

- 0xc, 0 – 001100

- tag 0x0, index 01, offset 0x4

- hit, word fetched by first miss on block 1.
- \( 0x0, 0 - 000000 \)

- tag \( 0x0 \), index 00, offset \( 0x0 \)

- hit, word fetched by first miss on block 0.

- \( 0x20, 0 - 100000 \)

- tag \( 0x1 \), index 00, offset \( 0x0 \)

- miss, tag on block 0 does not match, fetches words \( 0x20 \) and \( 0x24 \).

- \( 0x4, 0 - 000100 \)

- tag \( 0x0 \), index 0, offset \( 0x4 \)

- miss, tag on block 0 does not match, fetches words \( 0x0 \) and \( 0x4 \) into block 0.
• Longer lines tend to reduce miss rate

• this is only true when the line does not get too large for the cache and starts causing conflict misses

• longer lines also cause an increase in the time to transfer a line to cache (so looking at miss rate only is misleading – consider time as well)

• Cache/processor mitigation – early restart of processor (as soon as requested word is available)

• Cache/memory mitigation – requested word first (cache line order of return is altered depending on requested word)
insert figure 7.12
Memory/Cache Organization

- one word wide memory, bus, cache, processor

- K word wide memory, bus, cache, MUXed processor

- one word wide bus, cache and processor, interleaved memory

- the last is a compromise that pays latency once and then streams the words to cache (like a pipeline)
insert figure 7.13