1. Improving Cache via Parallelism: Nonblocking Caches
   - Usually, during cache miss, cache waits for data to be read from the memory and stalls all further requests
   - Nonblocking caches continue to serve future requests that result in a hit while the data for the miss is being fetched – hit under one miss
   - Further optimization may allow for multiple outstanding misses (i.e. nonblocking cache would also serve future requests that result in a miss)
     - We could have multiple misses for the same block

Example
For the following cache which optimization gives better performance: 2-way set associativity or hit under one miss? Calculate this both for FP and for integer programs. Assume that FP miss rate is 11.4% with direct-mapped cache and 10.4% with set associative cache. Miss rate for integer programs is 7.4% with direct mapped cache and 6% with 2-way set associative cache. Miss penalty is 16 cycles. Assume that the average memory stall time is simply a product of miss rate and miss penalty.

2. Improving Cache via Parallelism: Hardware Prefetching
   - Prefetch items from the memory before they have been requested by the processor
   - Data can be placed into the cache or in some buffer
     - For instructions, processor typically fetches two blocks on a miss – the requested one and the next consecutive one
     - If prefetched data is used, this generates next prefetch request

3. Improving Cache via Parallelism: Compiler-Controlled Prefetching
   - Compiler can insert special instructions to request prefetching
     - Register prefetch request will ask for the data to be loaded into register
     - Cache prefetch request will ask for the data to be loaded into cache
     - Prefetch can be faulting or non-faulting
   - Prefetch only makes sense if processor does not stall while waiting for prefetched data
     - Write hint informs processor of the write miss that writes a whole block, to avoid unnecessary read

Improving Hit Time
   - Hit time is critical because it determines (limits) the cycle time of the processor
   - It is mostly taken for accessing the tag and comparing it to the block address

1. Improving Hit Time: Small and Simple Caches
   - For first level cache
     - Small so that it can fit on the same chip as the processor
     - Simple, direct-mapped, so that the block read can be overlapped with tag check
   - For second level cache
     - Keeping tag memory on chip and data off chip provides fast checks
2. Improving Hit Time: Avoiding Address Translation

- CPU generates requests for virtual addresses
  - Some of those are in main memory but the address must be translated from the virtual address to the physical address
  - To make common case fast, we can store virtual addresses in cache – virtual cache
  - Virtual cache removes address translation overhead but
    - May violate page-level protection
    - Has to be flushed on context switch
    - Sometimes programs use different virtual addresses for the same physical address
    - I/O uses physical addresses

3. Improving Hit Time: Pipelined Cache Access

- Pipeline cache access so that it can last multiple clock cycles
  - Now clock cycle can be small
  - But we have greater penalty for mispredicted branches and more stalls for RAW dependencies

4. Improving Hit Time: Trace Cache

- Trace cache collect temporal information about data accessed, then loads this data into cache block (instead of the physical block)
  - Branch prediction is folded into cache
  - This solves the problem of low cache utilization – because of branches only a small portion of traditional cache block is used

Main Memory Organizations for Improving Performance

- Main memory communicates with cache and I/O devices
  - Performance measures
    - Latency – how long it takes for the request to be answered
    - Bandwidth – how much data can be read/written at once
  - Rather than just making memory faster (which is hard) or increasing bandwidth, some organizations better address these issues

Base Case

- 4 clock cycles to send the address
- 56 clock cycles for access time per word
- 4 clock cycles to send a word of data
- Assume cache block is 4-word long, word is 8 B
  - Miss penalty = 256 cycles

1. Higher Bandwidth: Wider Main Memory

- Wider the memory, less accesses we need on a miss
  - In the previous example, if the memory were 2 words long we would need 128 cycles
- Cache width is the same as memory width
  - As CPU still accesses cache a word at a time, words have to go through a multiplexer so that the correct word gets selected
  - Second level cache can help
2. Higher Bandwidth: Simple Interleaved Memory

- Memory is organized into banks
- Interleaved memory organization makes use of that by allowing banks to be read at the same time, interleaving multiple reads
- Miss penalty would be 76 cycles
- Writes can also be interleaved
- If interleaving is done so that adjacent words are stored in different banks, this optimizes sequential memory accesses
- To see benefit we need more banks than cycles to access memory

Example

Block size is 1 word, as well as memory bus width. Miss rate is 3%, memory accesses per instruction 1.2, cache miss penalty 64 cycles and CPI of ideal cache is 2. Calculate performance with 2 and 4 words for a block size, with and without interleaving

3. Higher Bandwidth: Independent Memory Banks

- Memory is organized into banks, but not sequentially
- Each device, such as I/O and caches, will access one bank

Exercise 5.1

We are given two machines, A and B with same processor (2GHz) and main memory, CPI of 1, and penalty of 100ns. Writing a word to a memory takes 100ns, and writing a block takes 200ns. Cache A is 2-way set associative and has 32B blocks, it is write through and does not allocate a block on write miss. Cache B is direct-mapped and has 32B blocks, it is write back and allocates a block on a write miss.

How to write a benchmark so that A is better than B?
How to write a benchmark so that B is better than A?