Digital Design and Computer Architecture(252-0028-00L), Spring 2023

Optional HW 7: Memory Hierarchy, Caches, Prefetching, and Virtual Memory

Instructor: Prof. Onur Mutlu

TAs: Juan Gomez Luna, Mohammad Sadrosadati, Mohammed Alser, Ataberk Olgun, Giray Yaglikci, Can Firtina, Geraldo De Oliveira Junior, Rahul Bera, Konstantinos Kanellopoulos, Nika Mansouri Ghiasi, Nisa Bostancı, Rakesh Nadig, Joel Lindegger, İsmail Emir Yüksel, Haocong Luo, Yahya Can Tuğrul, Julien Eudine

Released: Friday, June 23, 2023

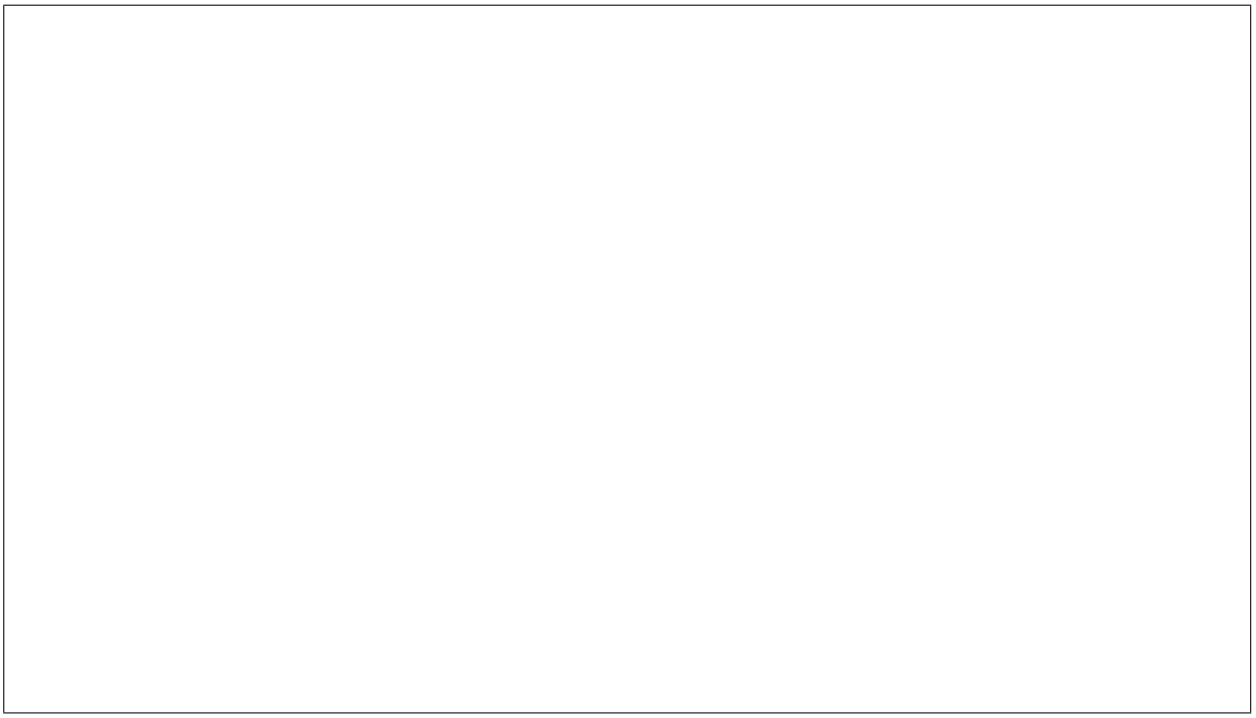
1 Instruction and Data Caches

Consider the following loop is executed on a system with a small instruction cache (I-cache) of size 16 B. The data cache (D-cache) is fully associative of size 1 KB. Both caches use 16-byte blocks. The instruction length and data word size are 4 B. The initial value of register `$1` is 40. The value of `$0` is 0. (Note: Assume that the first instruction of the loop is aligned to the beginning of a cache block).

```
Loop: lw   $6, X($1)
      addi $6, $6, 1
      sw   $6, Y($1)
      subi $1, $1, 4
      beq $1, $0, Exit
      j    Loop
Exit:  ...
```

(a) Compute I-cache and D-cache miss rates, considering:

- `X` and `Y` are different arrays.
- `X` and `Y` are the same array.



(b) Compute the average number of cycles per instruction (CPI), using a baseline ideal CPI (ideal caches) equal to 2, and a miss latency equal to 10 clock cycles.

(c) A compiler could unroll this loop for optimization. How would this affect CPI?

(d) How would the result of part (a) change with a 32-byte I-cache?

2 Reverse Engineering Caches I

You're trying to reverse-engineer the characteristics of a cache in a system so that you can design a more efficient, machine-specific implementation of an algorithm you're working on. To do so, you've come up with four patterns that access various *bytes* in the system in an attempt to determine the following four cache characteristics:

- Cache block size (8, 16, 32, 64, or 128 B)
- Cache associativity (2-, 4-, or 8-way)
- Cache size (4 or 8 KB)
- Cache replacement policy (LRU or FIFO)

However, the only statistic that you can collect on this system is cache hit rate after performing the access pattern. Here is what you observe:

Access Pattern	Addresses Accessed (Oldest → Youngest)								Hit Rate
A	0	4096	8192	12288	16384	4096	0		1/7
B	0	1024	2048	3072	4096	5120	6144	3072	0	1/9
C	0	4	8	16	32	64	128	256	512	4/9
D	128	1152	2176	3200	128	4224	1152			2/7

Based on what you observe, what are the following characteristics of the cache? (Be sure to justify clearly your answer for full credit.)

(a) Cache block size (8, 16, 32, 64, or 128 B)?

(b) Cache associativity (2-, 4-, or 8-way)?

(c) Cache size (4 or 8 KB)?

(d) Cache replacement policy (LRU or FIFO)?

3 Tracing the Cache

Assume you have three toy CPUs: 6808-D, 6808-T, and 6808-F. All three CPUs feature one level of cache. The cache size is 128 bytes, the cache block size is 32 bytes, and the cache uses LRU replacement. The only difference between the three CPUs is the associativity of the cache:

- 6808-D uses a direct mapped cache.
- 6808-T uses a two-way associative cache.
- 6808-F uses a fully associative cache.

You run the SPECMem3000 program to evaluate the CPUs. This benchmark program tests only memory read performance by issuing read requests to the cache. Assume that the cache is empty before you run the benchmark. The cache accesses generated by the program are as follows, in order of access from left to right:

`A, B, A, H, B, G, H, H, A, E, H, D, H, G, C, C, G, C, A, B, H, D, E, C, C, B, A, D, E, F`

Each letter represents a unique cache block. **All 8 cache blocks are contiguous in memory.** However, the ordering of the letters does not necessarily correspond to the ordering of the cache blocks in memory. For 6808-D, you observe the following cache misses in order of generation:

`A, B, A, H, B, G, A, E, D, H, C, G, C, B, D, A, F`

(a) By using the above trace, please identify which cache blocks are in the same set for the 6808-D processor. Please be clear.

(b) Please write down the sequence of cache misses for the 6808-F processor in their order of generation. (Hint: You might want to write down the cache state after each request).

(c) For 6808-T, you observed the following five cache misses in order of generation:

```
A, B, H, G, E
```

But, unfortunately, your evaluation setup broke before you could observe all cache misses for the 6808-T. Using the given information, which cache blocks are in the same set for the 6808-T processor?

(d) Please write down the sequence of cache misses for the 6808-T processor in their order of generation.

(e) What is the cache miss rate for each processor?

6808-D:

6808-T:

6808-F:

4 Memory Hierarchy

An enterprising computer architect is building a new machine for high-frequency stock trading and needs to choose a CPU. She will need to optimize her setup for *memory access latency* in order to gain a competitive edge in the market. She is considering two different prototype enthusiast CPUs that advertise high memory performance:

(A) Dragonfire-980 Hyper-Z

(B) Peregrine G-Class XTreme

She needs to characterize these CPUs to select the best one, and she knows from Prof. Mutlu's course that she is capable of reverse-engineering everything she needs to know. Unfortunately, these CPUs are not yet publicly available, and their exact specifications are unavailable. Luckily, important documents were recently leaked, claiming that all three CPUs have:

- Exactly 1 high-performance core
- LRU replacement policies (for any set-associative caches)
- Inclusive caching (i.e., data in a given cache level is present upward throughout the memory hierarchy. For example, if a cache line is present in L1, the cache line is also present in L2 and L3 if available.)
- Constant-latency memory structures (i.e., an access to any part of a given memory structure takes the same amount of time)
- Cache line, size, and associativity are all size aligned to powers of two

Being an ingenious engineer, she devises the following simple application in order to extract all of the information she needs to know. The application uses a high-resolution timer to measure the amount of time it takes to read data from memory with a specific pattern parameterized by *STRIDE* and *MAX_ADDRESS*:

```
start_timer()
repeat N times:
       memory_address <- random_data()
       READ[(memory_address * STRIDE) % MAX_ADDRESS]
end_timer()
```

Assume 1) this code runs for a long time, so all memory structures are fully warmed up, i.e., repeatedly accessed data is already cached, and 2) N is large enough such that the timer captures ***only*** *steady-state information.*

By sweeping *STRIDE* and *MAX_ADDRESS*, the computer architect can glean information about the various memory structures in each CPU.

She produces Figure 1 for CPU A and Figure 2 for CPU B.

Your task: Using the data from the graphs, reverse-engineer the following system parameters. If the parameter *does not make sense* (e.g., L3 cache in a 2-cache system), mark the box with an "X". If the graphs provide *insufficient information* to ascertain a desired parameter, simply mark it as "N/A".

(a) Fill in the blanks for Dragonfire-980 Hyper-Z.

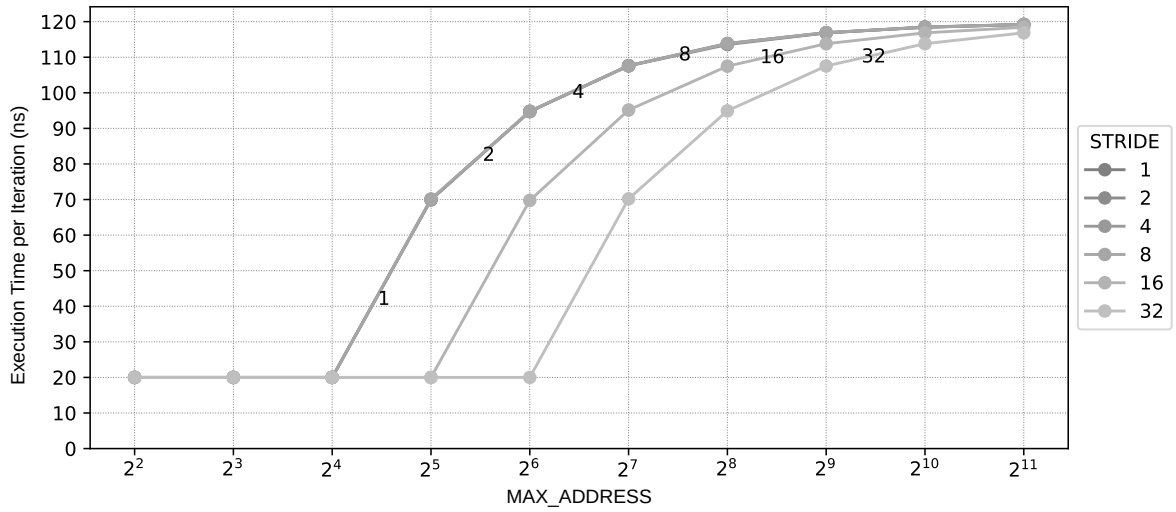


Figure 1: Execution time of the test code on CPU A for various values of *STRIDE* and *MAX_ADDRESS*. *STRIDE* values are labeled on curves themselves for clarity. Note that the curves for strides 1, 2, 4, and 8 overlap in the figure.

Table 1: Fill in the following table for CPU A (Dragonfire-980 Hyper-Z)

System Parameter	**CPU A: Dragonfire-980 Hyper-Z**			
	L1	L2	L3	DRAM
Cache Line Size (B)				
Cache Associativity				
Total Cache Size (B)				
Access Latency (ns) [1]				

[1] e.g., DRAM access latency means the latency of fetching the data from DRAM to L3, *not* the latency of bringing the data from the DRAM all the way down to the CPU. Similarly, L3 access latency means the latency of fetching the data from L3 to L2. L1 access latency is the latency to bring the data to the CPU from the L1 cache.

(b) Fill in the blanks for Peregrine G-Class XTreme.

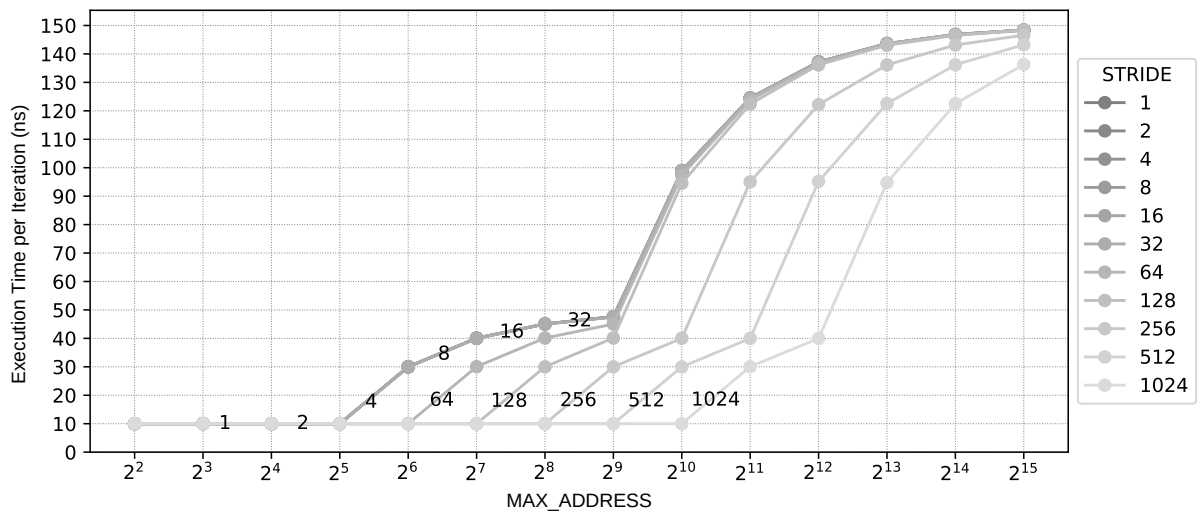


Figure 2: Execution time of the test code on CPU B for various values of *STRIDE* and *MAX_ADDRESS*. *STRIDE* values are labeled on curves themselves for clarity. Note that the curves for strides 1, 2, 4, 8, 16, and 32 overlap in the figure.

Table 2: Fill in the following table for CPU B (Peregrine G-Class XTreme)

System Parameter	**CPU B: Peregrine G-Class XTreme**			
	L1	L2	L3	DRAM
Cache Line Size (B)				
Cache Associativity				
Total Cache Size (B)				
Access Latency (ns) [1]				

[1] e.g., DRAM access latency means the latency of fetching the data from DRAM to L3, *not* the latency of bringing the data from the DRAM all the way down to the CPU. Similarly, L3 access latency means the latency of fetching the data from L3 to L2. L1 access latency is the latency to bring the data to the CPU from the L1 cache.

5 Virtual Memory

An ISA supports an 8-bit, byte-addressable virtual address space. The corresponding physical memory has only 128 bytes. Each page contains 16 bytes. A simple, one-level translation scheme is used and the page table resides in physical memory. The initial contents of the frames of physical memory are shown below.

Frame Number	Frame Contents
0	Empty
1	Page 13
2	Page 5
3	Page 2
4	Empty
5	Page 0
6	Empty
7	Page Table

A three-entry translation lookaside buffer that uses Least Recently-Used (LRU) replacement is added to this system. Initially, this TLB contains the entries for pages 0, 2, and 13. For the following sequence of references, put a circle around those that generate a TLB hit and put a rectangle around those that generate a page fault. What is the hit rate of the TLB for this sequence of references? (Note: LRU policy is used to select pages for replacement in physical memory.)

References (to pages): 0, 13, 5, 2, 14, 14, 13, 6, 6, 13, 15, 14, 15, 13, 4, 3.

(a) At the end of this sequence, what three entries are contained in the TLB?

(b) What are the contents of the 8 physical frames?

6 Virtual Memory and Caching I

A 2-way set associative write back cache with perfect LRU replacement requires 15×2^9 bits of storage to implement its tag store (including bits for valid, dirty and LRU). The cache is virtually indexed, physically tagged. The virtual address space is 1 MB, page size is 2 KB, cache block size is 8 bytes.

(a) What is the size of the data store of the cache in bytes?

(b) How many bits of the virtual index come from the virtual page number?

(c) What is the physical address space of this memory system?

7 Prefetching I

An architect is designing the prefetch engine for his machine. He first runs two applications A and B on the machine, with a stride prefetcher.

Application A:

```
uint8_t a[1000];
sum = 0;
for (i = 0; i < 1000; i += 4) {
    sum += a[i];
}
```

Application B:

```
uint8_t a[1000];
sum = 0;
for (i = 1; i < 1000; i *= 4) {
    sum += a[i];
}
```

`i` and `sum` are in registers, while the array `a` is in memory. A cache block is 4 bytes in size.

(a) What is the prefetch accuracy and coverage for applications A and B using a stride prefetcher. This stride prefetcher detects the stride between two consecutive memory accesses and prefetches the cache block at this stride distance from the currently accessed block.

(b) Suggest a prefetcher that would provide better accuracy and coverage for
i) application A?

ii) application B?

(c) Would you suggest using runahead execution for
i) application A. Why or why not?

ii) application B. Why or why not?

8 Prefetching II

Qualtel is designing a next-gen low-power mobile processor codenamed Nemo. You and your colleagues are tasked with designing the prefetcher for Nemo. Nemo has a single core, one level of cache, and a DRAM-based main memory system.

You need to examine different prefetcher designs and analyze the trade-offs involved.

- For all parts of this question, you need to compute the *coverage*, *accuracy* and *bandwidth* over of the prefetcher in its **steady state**.
- If there is a request to a cache block that has gone to main memory, a new request for the same cache block will not go to main memory as the outstanding request has not yet completed. Instead the new request will be coalesced with already outstanding request.

You run an application `libclassical` that has the following memory access pattern (note that these are cache block addresses):

$$A,\ A+1,\ A+2,\ A+7,\ A+8,\ A+9,\ A+14,\ A+15,\ A+16,\ A+21,\ A+22,\ A+23,\ ...$$

Assume this pattern continues for a long time.

(a) You first design a stride prefetcher that observes the last three cache block requests. If there is a constant stride S between the last three requests, the prefetcher issues a prefetch to the next cache block using the stride S. In absence of a constant stride, the prefetcher refrains from prefetching. What is the coverage of your stride prefetcher for `libclassical`? Show your work. Prefetcher coverage is defined as

$$\frac{Total\ number\ of\ correctly\ predicted\ prefetch\ requests}{Total\ number\ of\ unique\ cache\ block\ requests\ without\ the\ prefetcher}$$

(b) What is the the accuracy of your stride prefetcher for `libclassical`? Show your work. Prefetcher accuracy is defined as

$$\frac{Total\ number\ of\ correctly\ predicted\ prefetch\ requests}{Total\ number\ of\ prefetched\ requests}$$

(c) Your colleague designs a new prefetcher that, on a cache block access, prefetches the next N cache blocks. The coverage and accuracy of this prefetcher are 66.67% and 50% respectively for `libclassical`. What is the value of N? Show your work.

(d) The bandwidth overhead of the prefetcher can be defined as

$$\frac{Total\ number\ of\ unique\ cache\ block\ requests\ with\ the\ prefetcher}{Total\ number\ of\ unique\ cache\ block\ requests\ without\ the\ prefetcher}$$

What is the bandwidth overhead of this next-N-block prefetcher for `libclassical`? Show your work.

(e) What is the minimum value of N required to achieve a 100% prefetch coverage for `libclassical`? Show your work. Remember that you should consider the prefetcher's coverage in its steady state.

(f) What is the bandwidth overhead at this value of N? Show your work.

(g) However, you are not happy with the bandwidth overhead required to achieve a prefetch coverage of 100% with a next-N-block prefetcher. You aim to design a prefetcher that achieves a coverage of 100% with a 1× bandwidth overhead. Propose a prefetcher design that accomplishes this goal. Be concrete and clear.

Extra Exercises for Practicing

The following exercises are old exam questions that are conceptually similar to the ones above, but with slight alterations. We do not expect or recommend you to solve all of them, unless you think you are struggling with a particular concept, or would like to do practice runs on these old exam questions.

9 Cache Structure (Extra)

A byte-addressable system with 16-bit addresses ships with a two-way set associative, writeback cache with perfect LRU replacement. The tag store (including the tag and all other meta-data) requires a total of 4352 bits of storage. What is the block size of the cache? Assume that the LRU information is maintained on a per-set basis as a single bit. (Hint: $4352 = 2^{12} + 2^8$.)

10 Reverse Engineering Caches II (Extra)

Below, we have given you four different sequences of addresses generated by a program running on a processor with a data cache. Cache hit ratio for each sequence is also shown below. Assuming that the cache is initially empty at the beginning of each sequence, find out the following parameters of the processor's data cache:

- Associativity (1, 2 or 4 ways)
- Block size (1, 2, 4, 8, 16, or 32 bytes)
- Total cache size (256 B, or 512 B)
- Replacement policy (LRU or FIFO)

Assumptions: all memory accesses are one byte accesses. All addresses are byte addresses.

Sequence No.	Address Sequence	Hit Ratio
1	0, 2, 4, 8, 16, 32	0.33
2	0, 512, 1024, 1536, 2048, 1536, 1024, 512, 0	0.33
3	0, 64, 128, 256, 512, 256, 128, 64, 0	0.33
4	0, 512, 1024, 0, 1536, 0, 2048, 512	0.25

11 Cache Performance Analysis (Extra)

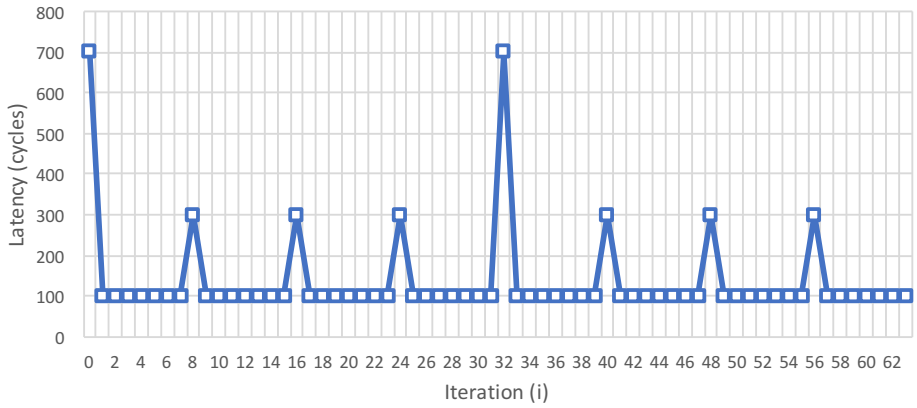
We are going to microbenchmark the cache hierarchy of a computer with the following two codes. The array `data` contains 32-bit unsigned integer values. For simplicity, we consider that accesses to the array `latency` bypass all caches (i.e., `latency` is *not* cached). `timer()` returns a timestamp in cycles.

```
(1) j = 0;
    for (i=0; i<size; i+=stride){
       start = timer();
       d = data[i];
       stop = timer();
       latency[j++] = stop - start;
    }

(2) for (i=0; i<size1; i+=stride1){
       d = data[i];
    }
    j = 0;
    for (i=0; i<size2; i+=stride2){
       start = timer();
       d = data[i];
       stop = timer();
       latency[j++] = stop - start;
    }
```

The cache hierarchy has two levels. L1 is a 4kB set associative cache.

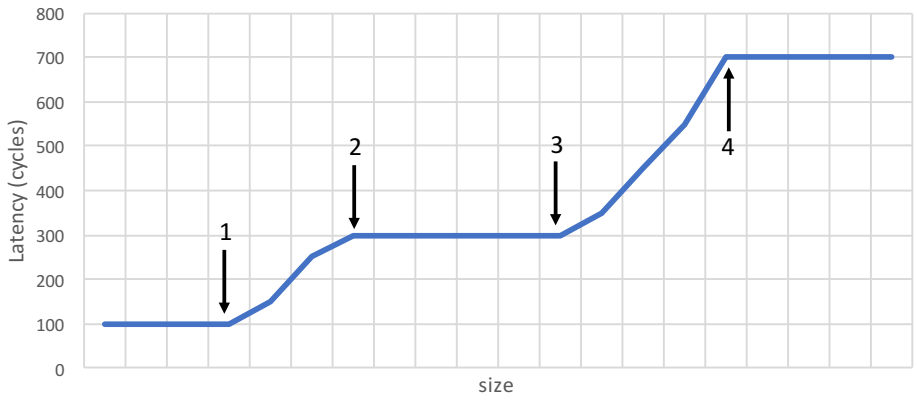
(a) When we run code (1), we obtain the latency values in the following chart for the first 64 reads to the array `data` (in the first 64 iterations of the loop) with `stride` equal to 1. What are the cache block sizes in L1 and L2?



(b) Using code (2) with `stride1` = `stride2` = 32, `size1` = 1056, and `size2` = 1024, we observe `latency[0]` = 300 cycles. However, if `size1` = 1024, `latency[0]` = 100 cycles. What is the maximum number of ways in L1? (Note: The replacement policy can be either FIFO or LRU).

(c) We want to find out the exact replacement policy, assuming that the associativity is the maximum obtained in part (b). We first run code (2) with `stride1` = 32, `size1` = 1024, `stride2` = 64, and `size2` = 1056. Then (after resetting `j`), we run code (1) with `stride` = 32 and `size` = 1024. We observe `latency[1]` = 100 cycles. What is the replacement policy? Explain. (Hint: The replacement policy can be either FIFO or LRU. You need to find the correct one and explain).

(d) Now we carry out two consecutive runs of code (1) for different values of `size`. In the first run, `stride` is equal to 1. In the second run, `stride` is equal to 16. We ignore the latency results of the first run, and average the latency results of the second run. We obtain the following graph. What do the four parts shown with the arrows represent?



Before arrow 1:

Between arrow 1 and arrow 2:

Between arrow 2 and arrow 3:

Between arrow 3 and arrow 4:

After arrow 4:

Explain as needed (if you need more):

12 Reverse Engineering Caches III (Extra)

You are trying to reverse-engineer the characteristics of a cache in a system, so that you can design a more efficient, machine-specific implementation of an algorithm you are working on. To do so, you have come up with three sequences of memory accesses to various *bytes* in the system in an attempt to determine the following four cache characteristics:

- Cache block size (8, 16, 32, 64, or 128 B).
- Cache associativity (2-, 4-, or 8-way).
- Cache replacement policy (LRU or FIFO).
- Cache size (4 or 8 KiB).

The only statistic that you can collect on this system is *cache hit rate* after performing each sequence of memory accesses. Here is what you observe:

Sequence	Addresses Accessed (Oldest → Youngest)								Hit Rate
1.	0	16	24	25	1024	255	1100	305	2/8
2.	31	65536	65537	131072	262144	8	305	1060	3/8
3.	262145	65536	4						2/3

Assume that the cache is initially empty at the beginning of the first sequence, but *not* at the beginning of the second and third sequence. The sequences are executed back-to-back, i.e., no other accesses take place in between sequences. Thus, **at the beginning of the second sequence, the contents are the same as at the end of the first sequence. At the beginning of the third sequence, the contents are the same as at the end of the second sequence**.

Based on what you observe, what are the following characteristics of the cache? Explain to get points.

(a) [20 points] Cache block size (8, 16, 32, 64, or 128 B)?

(b) [20 points] Cache associativity (2-, 4-, or 8-way)?

(c) [20 points] Cache replacement policy (LRU or FIFO)?

(d) [10 points] To identify the cache size (4 or 8KiB), you can access two addresses right after sequence 3 (i.e., the contents are the same as at the end of the third sequence) and measure the cache hit rate. Which two addresses would you choose? Explain your answer (there may be several correct answers).

13 Reverse Engineering Caches IV (Extra)

You are trying to reverse-engineer the characteristics of a cache in a system, so that you can design a more efficient, machine-specific implementation of an algorithm you are working on. To do so, you have come up with three patterns that access various *bytes* in the system in an attempt to determine the following four cache characteristics:

- Cache block size (8, 16, 32, 64, or 128 B)
- Cache associativity (1-, 2-, 4-, or 8-way)
- Cache size (4 or 8 KB)
- Cache replacement policy (LRU or FIFO)

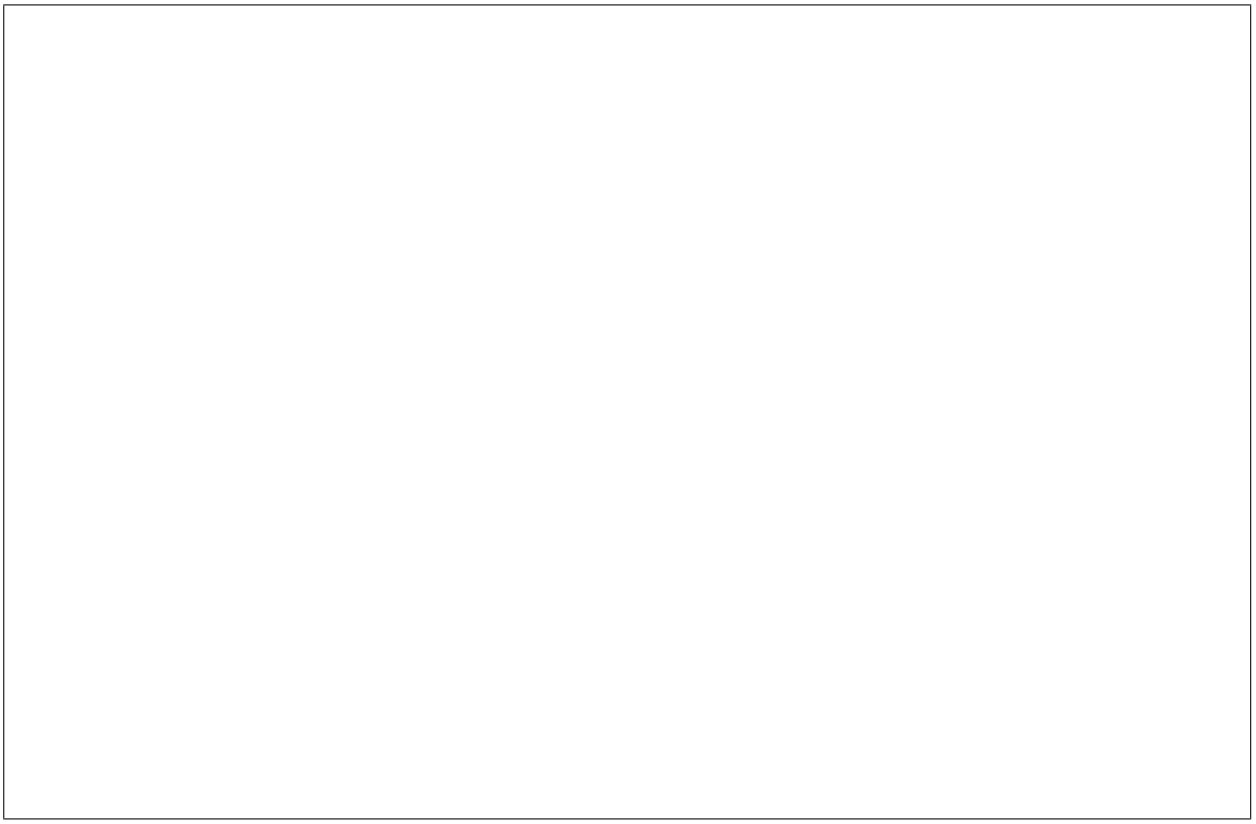
However, the only statistic that you can collect on this system is *cache hit rate* after performing the access pattern. Here is what you observe:

Sequence	Addresses Accessed (Oldest → Youngest)								Hit Rate
1.	0	4	8	16	64	128			1/2
2.	31	8192	63	16384	4096	8192	64	16384	5/8
3.	32768	0	129	1024	3072	8192			1/3

Assume that the cache is initially empty at the beginning of the first sequence, but not at the beginning of the second and third sequences. The sequences are executed back-to-back, i.e., no other accesses take place between the three sequences. Thus, **at the beginning of the second (third) sequence, the contents are the same as at the end of the first (second) sequence**.

Based on what you observe, what are the following characteristics of the cache?

(a) Cache block size (8, 16, 32, 64, or 128 B)?



(b) Cache associativity (1-, 2-, 4-, or 8-way)?

(c) Cache size (4 or 8 KB)?

(d) Cache replacement policy (LRU or FIFO)?

14 Virtual Memory and Caching II (Extra)

A four-way set-associative writeback cache has a 2^{11} * 89-bit tag store. The cache uses a custom replacement policy that requires 9 bits per set. The cache block size is 64 bytes. The cache is virtually-indexed and physically-tagged. Data from a given physical address can be present in up to eight different sets in the cache. The system uses hierarchical page tables with two levels. Each level of the page table contains 1024 entries. A page table may be larger or smaller than one page. The TLB contains 64 entries.

(a) How many bits of the virtual address are used to choose a set in the cache?

(b) What is the size of the cache data store?

(c) How many bits in the Physical Frame Number must overlap with the set index bits in the virtual address?

(d) On the following blank figure representing a virtual address, draw in bitfields and label bit positions for *cache block offset* and *set number*. Be complete, showing the beginning and ending bits of each field.

Virtual Address:

(e) On the following blank figure representing a physical address, draw in bitfields and label bit positions for *physical frame number* and *page offset*. Be complete, showing the beginning and ending bits of each field.

Physical Address:

(f) What is the page size?

(g) What is the size of the virtual address space?

(h) What is the size of the physical address space?

15 Prefetching III (Extra)

A runahead execution processor is designed with an unintended hardware bug: every other instruction in runahead mode is dropped by the processor after the fetch stage. Recall that the runahead mode is the speculative processing mode where the processor executes instructions solely to generate prefetch requests. All other behavior of the runahead mode is exactly as we described in lectures. When a program is executed, which of the following scenarios could happen compared to a runahead processor without the hardware bug and why? Circle YES if there is a possibility to observe the described behavior and explain in the box (either if you answer YES or NO). Assume that the program has no bug in it and executes correctly on the processor without the hardware bug.

(a) [8 points] The buggy runahead processor finishes the program *correctly* and *faster* than the non-buggy runahead processor. Why?

YES *NO*

(b) [8 points] The buggy runahead processor finishes the program *correctly* and *slower* than the non-buggy runahead processor. Why?

YES *NO*

(c) [9 points] The buggy runahead processor executes the program *incorrectly*. Why?

YES *NO*