Midterm 1

There are 8 questions with 16 parts worth 34 total points.

Name: Key

Computing ID: Key

- Time limit: 75 minutes.
- No notes, books, or calculators. Scratch paper is allowed.
- Because the test is given in multiple locations, we cannot answer questions about the test during the test. If you find a question ambiguous or confusing, explain that on the page near the question and it will be considered during grading.
- Grading occurs in gradescope, which shows one question to the grader at a time. Keep your comments and answers near the question they apply to.
- You may use pencil or pen, and may mark answers in any way a human grader can easily understand.
- Please write clearly. Ambiguous answers, such as `Ŧ` when `T` or `F` was expected, are graded as wrong.
- Partial credit is awarded for answers that demonstrate partial understanding.
- Grading is not linear: earning 50% of the available points may result in more than a 50% grade on the test.
- Unless otherwise specified in a specific question, assume that `int` is 32-bits, `long` and all pointers are 64-bits, and memory is little-endian.
- For your reference:
  - 2^{10} is abbreviated "K" or "Ki" (kilo)
  - 2^{20} is abbreviated "M" or "Mi" (mega)
  - 2^{30} is abbreviated "G" or "Gi" (giga)
  - 2^{40} is abbreviated "T" or "Ti" (tera)
  - 2^{50} is abbreviated "P" or "Pi" (peta)
  - 2^{60} is abbreviated "E" or "Ei" (exa)
  - byte or octet is abbreviated "B"
  - a "cold cache" is one with no data (i.e. all lines are invalid)
  - in assembly,
    - the order is `operation, source, destination`
    - `123(%rax, %rbx, 8)` means the memory address `123 + rax + (8 * rbx)`
    - there are has three shift operations: `sar` for signed `>>=`, `shr` for unsigned `>>=`, and `shl` for `<<=`; they have 64-, 32-, 16- and 8-bit versions like other operations (`shlq`, `shll`, and so on)
    - `cmp X,Y` sets the flags as if comparing the result of `Y -= X` to 0
    - `jg` and `jl` do signed-integer comparisons; `ja` and `jb` do unsigned-integer comparisons

Question 1
Consider a set-associative cache with

- 16-bit addresses
- 16-byte blocks
- 16 sets
- 2 lines per set, with LRU replacement
- write-back policy (also write-allocate)

Parts **b** and **c** below refer to the state of the cache after the memory accesses given in part **a**.

1.a. (4 pt) Starting from a cold cache, suppose the following addresses are accessed in the order provided. Select the accesses that are **hits** in the cache.

Assume all addresses are present in virtual memory (i.e., no page faults).

Select all that apply

- ☐ `0x3330` read
- ☐ `0x4330` write
- ✓ `0x333A` write
- ✓ `0x4330` read
- ☐ `0x5330` read
- ☐ `0x3330` write
- ✓ `0x333A` read
- ☐ `0x3320` read
- ☐ `0x3310` read
- ✓ `0x3330` read

1.b. (2 pt) Assume this is the only cache in the cache hierarchy. How many write operations are sent to RAM during the above accesses? Assume RAM can accept an entire block in a single write operation.

Answer: 2

1.c. (4 pt) For each valid line in the cache at the end of these accesses, list its set index (SI, a number), dirty bit (D, 0 or 1), and tag (a hexadecimal value) below.

To simplify grading, we'd prefer to see lines in ascending order of set index, but that is not required for full credit.

There are more lines below than you will need.

SI: 1	D: 0	tag: 33
SI: 2	D: 0	tag: 33
SI: 3	D: 1	tag: 33
SI: 3	D: 0	tag: 53
SI:	D:	tag:
SI:	D:	tag:
SI:	D:	tag:

1.d. (2 pt) What is the **size** of this cache?

Answer in abbreviated byte notation, like "16B" if the answer is 16 bytes, "256GB" if the answer is 256 gigabytes, etc.

Answer: 512B

Question 2
Consider a program that operates on a large 2D array of floating-point numbers in a way that accesses each number repeatedly. Program *D* uses `doubles` (64-bit floats) and program *F* uses `floats` (32-bit floats), but otherwise they are identical.

Assume the same cache is used for both programs and that each cache block are significantly larger than either a `float` or `double` value.

2.a. (1 pt) Consider the **spatial locality** exhibited by the two programs (i.e., the expected rate of cache hits because an accessed value is on the same block as a different value accessed earlier).

Pick One

- [✓ program *F* exhibits more spatial locality]
- program *D* exhibits more spatial locality
- they exhibit about the same spatial locality

2.b. (1 pt) Consider the **temporal locality** exhibited by the two programs (i.e., the expected rate of cache hits because the same address is accessed repeatedly with few other addresses in between).

Pick One

- program *F* exhibits more temporal locality
- program *D* exhibits more temporal locality
- [✓ they exhibit about the same temporal locality]

Question 3 (2 pt)
Write a C expression that extracts the bit in the 2^7's place from signed 32-bit integer `x`. For example, given `x = 0xF7F` it should produce the value `0` and given `x = 0x080` it should produce the value `1`.

Answer: (x>>7) & 1

Question 4
Consider the following assembly snippet, which uses the AT&T-syntax x86-86 assembly notation discussed in class and the textbook.

```
    movq    $0, %rcx            // line 1
start_loop:                     // line 2
    movl    %edi, %edx          // line 3
    andl    $15, %edx           // line 4
    movl    %edx, (%rax,%rcx,4) // line 5
    sarq    $4, %rdi            // line 6
    addq    $1, %rcx            // line 7
    cmpq    $16, %rcx           // line 8
    jne start_loop              // line 9
```

4.a. (4 pt) Assuming:

- `%rdi` represents a `long` named `x`
- `%rax` represents an `int*` (pointer to `int`) named `array`
- `%rcx` represents a `long` named `i`

write C code or pseudocode using a `while` or `for` loop that is equivalent to the assembly loop:

```
for(i = 0; i != 16; i+=1) {
    array[i] = x & 15
    x >>= 4
}
```

4.b. (2 pt) Which (if any) of the following changes would **not** change the values the loop writes to memory?
Select all that apply

- [✓] replacing line 1 with `xorl %ecx, %ecx`
- [✓] replacing line 3 with `movl $15, %edx` and line 4 with `andl %edi, %edx`
- [ ] replacing line 9 with `jg start_loop`
- [✓] replacing line 9 with `jl start_loop`

Question 5
Consider a virtual memory system with

- 32-bit virtual addresses (VA) stored in 4-byte pointers
- 4-byte page table entries (PTE)
- 38-bit physical addresses (PA)

Because physical addresses are larger than virtual addresses, each process in this system is limited to 4GB memory, but multiple processes running concurrently can have their full address space present in RAM at the same time.

5.a. (1 pt) How many bits is a physical page number (PPN) in this system?
Pick One

- ○ 32
- ○ 38
- ○ 32 − (bits per page offset)
- [✓ 38 − (bits per page offset)]
- ○ (32 − (bits per page offset)) / (number of levels)
- ○ (38 − (bits per page offset)) / (number of levels)
- ○ (32 − (bits per page offset)) / (number of levels) + 6

5.b. (2 pt) Suppose we want each array of PTEs to be exactly one page in size. Which of the following page sizes allow that given the VA, PTE, and PA sizes given above? For each option you pick, also state the number of levels that makes it work.
Select all that apply

- [✓] 256B, with levels = 4
- [✓] 4KB, with levels = 2
- [ ] 64KB, with levels = ____
- [✓] 128KB, with levels = 1
- [ ] 256KB, with levels = ____

Question 6
By design, operating systems and hardware work together to keep user-mode programs from knowing the contents of their page table. Suppose a security vulnerability allows a user-mode program to map its page table to virtual addresses, so that it can access page table entries using pointers. Assume this includes the entire page table, include the parts that describe kernel-only memory, but not the data pages that the page tables describe.

6.a. (1 pt) If a user-mode process figures out how to map its page-table pages into its address space as read-only memory, it gains which of the following powers?
Select all that apply

- ☐ it can install its own code as an exception handler
- [✓ it can jump to kernel code in user mode]
- ☐ it can modify kernel code and data
- ☐ it can read other processes' memory

6.b. (2 pt) If a user-mode process figures out how to map its page-table pages into its address space as read/write memory, it gains which of the following powers?
Select all that apply

- [✓ it can install its own code as an exception handler]
- [✓ it can jump to kernel code in user mode]
- [✓ it can modify kernel code and data]
- [✓ it can read other processes' memory]

Question 7 (2 pt)
Suppose a system has 2-byte PTEs arranged as follows (described twice, once as an image and once in text):

15 … 4	3 … 2	1	0
PPN	unused	W	P

- low-order bit is 1 if the page is present, 0 if not
- next-lowest-order bit is 1 if the page is writeable, 0 if not
- next-lowest-order 2 bits are unused
- highest-order 12 bits are a PPN

VPNs are 12 bits and the page table is two levels.
If the VPN is `0xACE` and the PTBR stores physical address `0x10000`, what is the physical address of (the first byte of) the first PTE accessed while translating this address?
Answer as a hexademical number, like `0x0` or `0xACE`

Answer: 0x10056

Question 8
Consider a virtual memory system where

- Addresses are 48-bits, made up of
  - a 36-bit VPN (broken into four 9-bit parts)
  - a 12-bit page offset
- There is a set-associative TLB

(aside: this is the same that the textbook described as a system Intel uses)

8.a. (2 pt) Starting from a cold TLB, a program makes two memory accesses, first to address *X* (which is a cold miss in the TLB) and then to address *Y*. The access to address *Y* will be a hit in the TLB if and only if
Pick One

- ○ *X* and *Y* have the same high-order VPN part
- ○ *X* and *Y* have the same low-order VPN part
- ○ *X* and *Y* have the same PPN
- [✓ *X* and *Y* have the same VPN]
- ○ *X* and *Y* use the same TLB set index
- ○ *X* and *Y* use the same TLB tag

8.b. (2 pt) Suppose I write a program that uses just four pages of data:

- Code, on page with VPN `0x008000000`
- Globals and heap, on page with VPN `0x010000000`
- Stack, on page with VPN `0x7FFFFFFFF`

Ignoring kernel-mode memory, but counting pages used to store arrays from the page table, how many total physical pages are used by this program?
Answer as a base-10 integer, like `0` or `256`

Answer: dropped because of typo in question text

Pledge:
On my honor, I pledge that I have neither given nor received help on this assignment.

Signature: key