Name:	UID:

For Discussion Credit, fill out following attendance sheet: https://tinyurl.com/cs33f25week10

1. Single Choice

For the following questions, select the best option:

a. Which of the following is the best justification for using the middle bits of an address as the set index into a cache rather than the most significant bits?

   (a) Indexing with the most significant bits would necessitate a smaller cache than is possible with middle-bit indexing, resulting in generally worse cache performance.

   (b) It is impossible to design a system that uses the most significant bits of an address as the set index.

   (c) The process of determining whether a cache access will result in a hit or a miss is faster using middle-bit indexing.

   (d) A program with good spatial locality is likely to make more efficient use of the cache with middle-bit indexing than with high-bit indexing.

b. When you print the address of a variable from C, what kind of address is that?

   (a) Local Address

   (b) Physical Address

   (c) Virtual Address

   (d) Home Address

c. For a floating point number, what would be an effect of allocating more bits to the exponent part by taking them from the fraction part?

   (a) You could represent fewer numbers, but they could be much larger.

   (b) You could represent the same numbers, but with more decimal places.

   (c) You could represent larger and small numbers, but with less precision.

   (d) Some previously representable numbers would now round to infinity.

d. The following code is parallelized with 4 threads. Does the following code snippet contain a race condition? If there is a race condition, what synchronization tool could remove the race condition? (assume the function is called in a similar to the thread lab)

```
int sum = 0;
int local_sum[4] = {0, 0, 0, 0}
double sum(double a[N], int thread_id) {
    for (int i = thread_id * (n/4); i < (thread_id + 1) * (n/4)) {
        local_sum[thread_id] += a[i];
    }

    if (thread_id == 0) {
        sum = local_sum[0] + local_sum[1]
              + local_sum[2] + local_sum[3];
    }
}
```

(a) There is no race condition

(b) Atomics

(c) Semaphore

(d) Mutex

(e) Barrier

2. Deadlock or Not?

Can the following program deadlock? Why or why not?

Initially: $a = 1, b = 1, c = 1$

Thread 1:	Thread 2:
P(a)	P(c)
P(b)	P(b)
V(b)	V(b)
P(c)	V(c)
V(c)	
V(a)	

3. Miss-ed Ya?

Consider a directed mapped cache of size 64K with block size of 16 bytes. Furthermore, the cache is write-back and write-allocate. You will calculate the miss rate for the following code using this cache. Remember that sizeof(int) == 4. Assume that the cache starts empty and that local variables and computations take place completely within the registers and do not spill onto the stack.

Now consider the following code to copy one matrix to another. Assume that the src matrix starts at address 0 and the dest matrix immediately follows it.

```
1 double copy_matrix(int dest[ROWS][COLS], int src[ROWS][COLS]) {
2     for (int i = 0; i < ROWS; i++) {
3         for (int j = 0; j < COLS; j++) {
4             dest[i][j] = src[i][j];
5         }
6     }
7 }
```

a. What is the cache miss rate if ROWS = 128 and COLS = 128?

   Miss Rate = _____ %

b. What is the cache miss rate if ROWS = 128 and COLS = 192

   Miss Rate = _____ %

c. What is the cache miss rate if ROWS = 128 and COLS = 256

   Miss Rate = _____ %

4. Stack Overflow

Alright, you're a hacker now and you happen to have an inside source at a company that can provide you with assembly for the company's administrative code. The code is typically air tight but your source tells you a rookie programmer, Alex, was just hired and that their code has vulnerabilities and no OS protections (Alex is not the best at their job). Specifically, your source provides you with this snippet of assembly that is in charge of taking in a user password attempt as well as checking whether the user has an existing password or not (why these two tasks are in one function is beyond me, but Alex is a rookie).

```
00000000086012b4 <get_attempt_and_check_null_password>:
                                  # first argument is user password
  86012b4:    48 83 ec 38       sub $0x38, %rsp
  86012b8:    48 89 fd          mov %rdi, %rbp
  86012bb:    48 89 e7          mov %rsp, %rdi
  86012be:    e8 38 02 00 00    callq 0x501e2b <Gets>    # looks familiar?
  86012c3:    48 89 ef          mov %rbp, %rdi
  86012c6:    e8 38 ff ff ff    callq 0x702ee3 <Null_Pw_Check> # important?
  86012cb:    b8 01 00 00 00    mov $0x1, %eax
  86012d0:    48 83 c4 38       add $0x38, %rsp
  86012d4:    c3                retq
```

Your inside source also informs you of the existence of the following function:

```
0000000004b023e4 <Print_String_Exit>:
  ...     # Prints whatever string is passed in
  ...     # as second arg and exits with value of
  ...     # first arg
```

Lastly, your source informs you that `rsp` will be set to

`0xFF FF FF FF 57 4E 3B 52`

when entering the snippet in the first image (this is a really good source).

a. **Assuming you acquired an individual's username information, what string will allow you to view their password?**

Exploit Code (7 bytes):

```
48 89 ee                mov %rbp, %rsi      # password to 2nd arg
48 31 ff                xor %rdi, %rdi      # exit code = 0
c3                      ret                 # return to address on stack
```

Sadly, before this code was able to be deployed Alex's supervisor saw it and angrily told Alex to fix it and explain to them that somebody could inject executable code onto the stack and cause problems, as well as judging Alex for creating such an impractical function. Alex, narrowing in on the term executable code (and not really listening to anything else that was said) simply turned on an OS feature that made the stack **non-executable**. Luckily for you, your inside source is a good one and has provided you with the following farm in order to circumnavigate this issue.

```
00000000a576f3e2 <good_function>:
  ...
  a576f420:      b8 48 89 fc 90      some instr
  a576f425:      c3                   retq

00000000e2e2e2e2 <some_function>:
  ...
  e2e2efff:     c7 09 07 48 89 ca     some instr
  e2e2e305:     c3                    retq

0000000042013122 <bad_function>:
  ...
  42013122:     c6 48 89 d6 20 c0     some instr
  42013128:     c3                    retq
```

```
00000000052ea100 <this_function>:
  ...
  52ea116:     8d 87 48 89 f9 90      some instr
  52ea11c:     c3                     retq

00000000c462a204 <that_function>:
  ...
  c462a24c:     c7 07 48 89 e6 90      some instr
  c462a212:     c3                     retq

00000000ffffff00 <f_function>:
  ...
  100000f0:    b8 48 89 fe 20 d9       some instr
  100000f6:    c3                      retq
```

mov Instructions:

Hex Bytes	Instruction
48 89 fc	mov %rdi, %rsp
48 89 f9	mov %rdi, %rcx
48 89 ca	mov %rcx, %rdx
48 89 d6	mov %rdx, %rsi
48 89 e6	mov %rsp, %rsi
48 89 fe	mov %rdi, %rsi

Other Instructions:

Hex Bytes	Instruction
90	nop
20 c0	and %al, %al
20 d9	and %bl, %cl

b. **What string will still allow you to view the user's password?**

Alright, at this point you are almost hit by a car and, having narrowly avoided death, decide that being a hacker is not the right thing to do. You turn a new leaf and want to help Alex before they are ultimately fired.

c. **What methods can you offer to Alex in order to prevent people from abusing the code (although you are not allowed to tell Alex to get rid of the impractical combination of getting input and checking password existence because that would hurt Alex's feelings)?**

5. Code Optimization/Performance

Suppose we wish to write a function to evaluate a polynomial, where a polynomial of degree n is defined to have a set of coefficients $a_0, a_1, a_2, \ldots, a_n$. For a value x, we evaluate the polynomial by computing

$$a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n.$$

This evaluation can be implemented by the following function, having as arguments an array of coefficients a, a value x, and the polynomial degree `degree` (the value n in Equation 5.2). In this function, we compute both the successive terms of the equation and the successive powers of x within a single loop:

```
1 double poly(double a[], double x, long degree) {
2     double result = 0;
3     double xpwr = 1;
4     for (long i = 0; i <= degree; i++) {
5         result += a[i] * xpwr;
6         xpwr = x * xpwr;
7     }
8     return result;
9 }
```

a. For degree n, how many additions and how many multiplications does this code perform?

b. On our reference machine, with arithmetic operations having the latencies shown in the figure below, we measure the CPE for this function to be 5.00. Explain how this CPE arises based on the data dependencies formed between iterations due to the operations implementing lines 7–8 of the function (code inside the `for` loop).

Operation	Integer			Floating point		
	Latency	Issue	Capacity	Latency	Issue	Capacity
Addition	1	1	4	3	1	1
Multiplication	3	1	1	5	1	2
Division	3–30	3–30	1	3–15	3–15	1

6. Memory Storage

Consider the following linked-list traversal function, where all linked_list items have been allocated dynamically (by calling malloc).

```
1  struct linked_list;
2
3  long long int total=10000;
4
5  typedef struct linked_list {
6    struct linked_list* next;
7    long long int value;
8  } linked_list;
9
10 long long int* traverse(linked_list* root) {
11   linked_list* current = root;
12   int total=0;
13
14   while(current->next) {
15     total += current->value;
16     current = current->next;
17   }
18   return &total;
19 }
```

Variables can be in global memory, the stack, the heap, text, external libraries, data, or in registers. What is the most likely location of each of the following variables? (be as specific as possible) (6pts)

(a) `total` on line 2: _____

(b) `root` on line 9: _____

(c) `*root` on line 9: _____

(d) `current` on line 15: _____

(e) `total` on line 17: _____

(f) `&total` on line 17: _____

(g) The assembly instructions: _____