Name: _____

Email: _____

Q1: _____/ 35

Q2: _____/ 35

Q3: _____/ 40

Q4: _____/ 25

Q5: _____/ 25

Q6: _____/ 20

Q7: _____/ 20

Total: _____ / 200

Question 1) Write, in x64 (x86 64 bit assembly) the following function:

```
// return the maximum element (not the index of the maximum element, but
// its value.
unsigned long long max(unsigned long long *array, int elements_in_array);
```

Question 2) Write, in x64 (64 bit x86 assembly) the following function. Be sure to follow all calling conventions and include all necessary pseudo directives.

```
char *find_char(char *s, char c) {
        if (!*s) return NULL;
        if (*s == c) return s;
        return find_char(s++, c);
}
```

Question 3) Write in C the function: int atoi(char *s); This routine converts ASCII strings of base 10 integers to binary integers. For example:

atoi(“1234”) = 1234
atoi(“-5678”) = -5678

Moreover, there are some unusual boundary cases:

atoi(“1234abcd”) = 1234

(As an aside, I am very serious -- atoi is a standard interview question. Regardless of this final exam, make sure you can do a quality job of this on a whiteboard!)

4) Write in C the following function:

```
#define BIG_ENDIAN 0
#define LITTLE_ENDIAN 1

int what_endian_is_this_machine(); // return 0 or 1
```

Question 5)

Consider a direct-mapped (non-associative) cache with a cache line size of 4 bytes and a total cache size of 16 bytes. For simplicity sake, let's assume we are working on an 8 bit system (so the addresses are relatively short). Show how the address will be divided into tag, index, and line size components:

7 6 5 4 3 2 1 0

Draw the cache and all of its rows and columns. Assume a write-back cache.

Now assume memory is initialized such that Memory[Address] = Address. Show the state of the cache after the following accesses have occurred (feel free to re-use your drawing above).

Read 0x01
Write 0x07, 6 // Memory[0x07] = 6
Read 0x06
Write 0x06, 7
Read 0x20

Finally, how many accesses in the above sequence were hits?

How many were misses?

Question 6) (Short answer)

1) Increasing the associativity of a cache reduces conflict misses. (T / F)

2) Give the number of sets of a cache of 512 bytes, 2-way associativity with a block size of 32 bytes.

3) List out the three functions provided by virtual memory.

4) Briefly describe temporal locality and spatial locality.

6) Briefly describe the potential risk of a double free.

7) Calculate the miss rate of the following code

```
int x[2][32], i, sum = 0;

for (i = 0; i < 32; i++)
    sum += x[0][i] * x[1][i];
```

Assumptions:
Initially empty cache of 128 bytes, direct-mapped and with 16 bytes cache blocks sizeof(int) = 4. Array x begin at memory address 0x0 and is stored in row-major order. All other variables besides entries of array x is stored in registers.

8) Which of the following statements, if any, are equivalent:

i)
```
int a[10];
  int *ip;
  ip = a + 3;
```

ii)
```
int a[10];
    int *ip = *a[0] + 3;
```

iii)
```
int a[10];
    int *ip = &a[3];
```

Question 7) (Short answer)

1) What is the difference between write-through and write-back?

2) (T / F) When malloc is called, a block of memory is allocated on the stack and the pointer to this block is returned.

3) (T / F) In user space, if dynamic memory becomes fragmented from a series of malloc and free calls, the allocated blocks can be moved and compacted to create larger contiguous free blocks.

4) (T / F) Free must only be given a pointer to a block previously allocated by malloc.

5) Assuming the following struct starts at address zero, what are the starting addresses for each field in the struct on a 64-bit system? (Or, how big is the size of test)

```
struct test{
    int a;
    char b;
    short c[5];
    int* d;
    int e;
}
```

6) For the following function, give the results of these function calls.

```
        switchtest(1,4);
        switchtest(3,2);
        switchtest(4,6);
        switchtest(2,3);

int switchtest(int a, int b) {
    int result = b;
    switch(a) {
        case 1: result = result << 2; break;
        case 3: result = result | 1; break;
        case 4:
        case 5: result = result * 2;
        case 6: result = result - 1; break;
        default: result = 12345;
    }
    return result;
}
```