Computer Systems Organization
CSCI-UA.0201 Fall 2019

Mid-Term Exam
ANSWERS

Write all answers on these exam sheets. If you need more space, write on the back.

1. True/False. Please circle the correct response. There are <u>no</u> trick questions.
   a. **F** Registers and Memory (RAM) are different names for the same thing.
   b. **T** In x86-64 assembly, an address (corresponding to a C pointer) is 64 bits.
   c. **T** Putting **`#include "foo.h"`** in your C file is the equivalent of typing the contents of foo.h into your file.
   d. **F** All real numbers can be represented exactly in IEEE floating point format.
   e. **T** $2^{25} = 32M$
   f. **F** The bitwise-and operator (**`&`** in C) is used to flip the bits of a variable.
   g. **F** In C, the statement, "**`s1 = s2;`**", where **`s1`** and **`s2`** are variables of type **`char *`**, copies the string pointed to by **`s2`** into the string pointed to by **`s1`**.
   h. **T** In Intel jargon, a "quadword" is a 64-bit quantity.
   i. **F** In x86-64 assembly, the **`addl`** instruction can add two 32-bit values in memory together.
   j. **T** The IEEE floating point representation does not use two's complement for negative numbers.

2. For each code snippet, below, indicate what the result of executing the code is. If the code won't compile, indicate so.

   a.
   ```
   int *p = 0;
   int x = *p;
   printf("%d\n", x);
   ```
   Answer: <u>Segmentation Fault</u>

   b.
   ```
   char s1[6] = "Above";
   char *p1 = s1;
   char s2[6] =  "Below";
   char *p2 = s2+4;
   while(*p1) *(p2--) = *(p1++);
   printf("%s\n", s2);
   ```
   Answer: <u>evobA</u>

   c.
   ```
   int x = 5;
   printf("%d\n", (~x)+2);
   ```
   Answer: <u>-4</u>

d. **`char c = 'H'; // 72 in ASCII`**
`printf("%c\n", c + (c >> 3));` Answer: Q (ASCII 72 + 72/8 = 81)

e. **`int x = 9;`**
`printf("%x\n", (x | (1 << 2))); // HEX!` Answer: d (1101 binary)

3. Multiply the following two binary numbers (without converting to decimal), and then show the result in binary and in hex. Show all work and write neatly.

```
      1011
    × 1010
         0
     1011
       0
   1011
   1101110
```

Result in binary: 1101110

Result in hex: 6E

4. Given the following declaration,

```
typedef struct cell {
    int val;
    struct cell *next;
} CELL;
```

in the space below write C code that constructs a linked list of ten **`CELL`**s, pointed to by the variable **`head`**, whose **`val`** fields contain the numbers 1 through 10 (in order, so that **`head`** points to the cell containing 1). This should take around 7 lines. Write neatly.

```
CELL *head = NULL;
for(int i=10; i>=1; i--) {
  CELL *p = (CELL *) malloc(sizeof(CELL));
  p->val = i;
  p->next = head;
  head = p;
}
```

5. Fill in the missing X86-64 assembly code, below, for a function **`largest()`** that takes two parameters – an integer array (in **`%rdi`**) and the size of the array (in **`%esi`**) – and returns (in **`%eax`**) the value of the largest element of the array. You can assume all the elements of the array are positive.

```
        // a[] in %rdi
        // size in %esi
        // %eax will keep track of largest value.
_largest:
        pushq   %rbp
        movq    %rsp,%rbp
        movl    $0,%eax                 #initialize %eax to 0.
        movq    $0,%rcx                 # i in %rcx
TOP:
        cmpl    %esi,%ecx               #compare i to size
        jge     DONE

        cmpl    %eax,(%rdi,%rcx,4)    # if a[i] <= %eax,
        jle     NEXT                  # then go to next element
        movl    (%rdi,%rcx,4),%eax
NEXT:
        incq    %rcx
        jmp     TOP
DONE:
        popq    %rbp
        ret
```

6. 32-bit IEEE floating numbers have one sign bit, 8 exponent bits (with a 127 bias), and 23 fraction bits.

   a. In order to operate on the individual bits of a **`float`** variable, it should be treated as an **`unsigned int`**, but not be converted or truncated by the compiler. Fill in the missing code below to do this.

```
void foo(float f)
{
  unsigned int x = *((unsigned int *) &f);
                                         // x now has the same
                                         // bit pattern as f.
```

   b. Define a preprocessor macro **`EXP`** for extracting the exponent bits of a floating point number after it has already been converted to an unsigned int. The exponent bits should end up in the rightmost bits of the result.

```
#define EXP(x)  ((x) >> 23) &0xff)
```

   c. What is the value (in decimal) of the IEEE floating point number whose fields (sign, exponent, and fraction) have the following bit patterns:

      sign = 1
      exponent = 10000001 (= 129)
      fraction = 11110000000000000000000

   Show your work.

   The sign is 1, so the number is negative.
   The actual exponent = stored exponent – bias = 129 – 127 = 2
   The mantissa is "1." followed by the fraction bits = 1. 1111 (ignoring trailing zeros)

   So the number is $-1.1111 \times 2^2 = -111.11 \times 2^0 = -(4 + 2 + 1 + ½ + ¼) = -7.75$

   Answer: -7.75