Number Representation & Strings

A. What is the value of the `signed char` **`0x9E`** in decimal?

B. What is the value of the `unsigned char` **`37`** in binary?

C. If **`a = 0x2C`**, complete the bitwise C statement so that **`b = 0x1F`.**

```
b = a____0x______
```

For the following problems we are working with a floating point representation that follows the same conventions as IEEE 754 except using 7 bits split into the following fields:

```
Sign (1) Exponent (3) Mantissa (3)
```

D. What is the magnitude of the bias of this new representation?

E. What is the decimal value encoded by **`0b1110101`** in this representation?

F. What value will be read after we try to store -18 in this representation? (Circle one)

-16 -NaN -∞ -18

For the following problem, assume we are working with C strings encoded in ASCII. Consider the declaration:

```
char str[] = "Hello!";
```

G. What will be stored in the array str?

Pointers & Memory

For this problem we are using a 64-bit x86-64 machine (**little endian**). The current state of memory (values in hex) is shown below:

```
char* charP = 0xD;
short* shortP = 0x1E;
```

Word Addr	+0	+1	+2	+3	+4	+5	+6	+7
0x00	20	F6	EF	EA	A2	5E	9F	1A
0x08	A2	D0	4F	C4	A0	0C	F7	27
0x10	B8	BD	1A	CA	35	95	CB	80
0x18	84	3F	02	4F	8E	F3	F6	E5
0x20	CD	4A	F6	48	1A	6F	7E	63

A. Using the values shown above, ill in the C type and hex value for each of the following C expressions. Leading zeros are not required for the hex values.

C Expression	C Type	Hex Value
`*(charP + 6)`		
`(int**)shortP - 2`		

B. For the following snippet of C code, draw out a box-and-arrow diagram for the allocated memory.

```
int x = 351, y = 332;
int *p = &x;
int **q = &p;
*q = &y;
*(*q) = x;
```

C & Assembly

Answer the questions below about the following x86-64 assembly function:

```
mystery:
            jmp     .L2                      # Line 1
.L4:        addq    $1, %rdi                 # Line 2
            movb    %al, (%rsi)              # Line 3
            leaq    1(%rsi), %rsi            # Line 4
.L2:        movzbl  (%rdi), %eax             # Line 5
            testb   %al, %al                 # Line 6
            je      .L3                      # Line 7
            cmpb    %dl, %al                 # Line 8
            jne     .L4                      # Line 9
.L3:        movb    $0, (%rsi)               # Line 10
            retq                             # Line 11
```

A. What **variable type** would `%rdi` be in the corresponding C program?

B. What **variable typ**e would the third argument be in the corresponding C program?

C. This function uses a `while` loop. Fill in the two conditionals below, using register names as variable names (no declarations necessary).

```
while (__________&&__________ )
```

D. Taking the variable types into account, describe at a high level what the *purpose* of Line 10 is (not just what it does mechanically).

E. Describe at a high level what you think this function *accomplishes* (not line-by-line).

Procedures & The Stack

The recursive function `count_nz` counts the number of *non-zero* elements in an `int` array. Example: if `int a[] = {-1,0,1,255}`, then `count_nz(a,4)` returns 3. The function and its x86-64 *disassembly* are shown below:

```
int count_nz(int* ar, int num) {
      if (num > 0)
           return !!(*ar) + count_nz(ar + 1,num - 1);
      return 0;
}
```

```
0000000000400536 <count_nz>:
 400536: 85 f6              testl %esi,%esi
 400538: 7e 1b              jle 400555 <count_nz+0x1f>
 40053a: 53                 pushq %rbx
 40053b: 8b 1f              movl  (%rdi),%ebx
 40053d: 83 ee 01           subl $0x1,%esi
 400540: 48 83 c7 04        addq $0x4,%rdi
 400544: e8 ed ff ff ff     callq 400536 <count_nz>
 400549: 85 db              testl %ebx,%ebx
 40054b: 0f 95 c2           setne %dl
 40054e: 0f b6 d2           movzbl %dl,%edx
 400551: 01 d0              addl %edx,%eax
 400553: eb 06              jmp 40055b <count_nz+0x25>
 400555: b8 00 00 00 00     movl $0x0,%eax
 40055a: c3                 retq
 40055b: 5b                 popq %rbx
 40055c: c3                 retq
```

A. How much space (in bytes) does this function take up in our final executable?

B. The compiler automatically creates labels it needs in assembly code. How many labels are used in `count_nz` (including the procedure itself)?

C. In terms of the *C function*, what value is being saved on the stack?

D. What is the return address to `count_nz` that gets stored on the stack (in hex)?

E. Assume `main` calls `count_nz(a,5)` with an appropriately-sized array and then prints the result using `printf`. Starting with (including) `main`, answer the following *in the number of stack frames*.

Total created:	Max depth:

F. Assume `main` calls `count_nz(a,6)` with **`int`** `a[] = {3,5,1,4,1,0}`. We find that the return address to `main` is stored on the stack at address `0x7fffeca3f748`. What data will be stored on the stack at address **`0x7fffeca3f720`**?

0x7fffeca3f748	<ret addr to main>
0x7fffeca3f740	
0x7fffeca3f738	
0x7fffeca3f730	
0x7fffeca3f728	
0x7fffeca3f720	

G. A similar function `count_z` that counts the number of *zero* elements in an array is made by making a single change to `count_nz`. What is the address of the changed assembly instruction?

Design Questions

A. What values can `S` take in an `x86-64` memory operand? *Briefly* describe why these choices are useful/important.

Values:
Importance:

B. Until very recently (Java 8/9), Java did not support *unsigned* integer data types. Name one advantage and one disadvantage to this decision to omit `unsigned`.

Advantage:
Disadvantage:

C. **Condition codes** are part of the *processor/CPU state*. Would our instruction set architecture (ISA) still work if we got rid of the condition codes? *Briefly* explain.

Circle one: Yes No
Explanation: