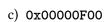# 1 Floating Point

1.1 Convert the following single-precision floating point numbers from hexadecimal to decimal or from decimal to hexadecimal using the IEEE 754 Floating Point Standard. You may leave your answer as an expression.

a) `8.25`

b) `39.5625`

c) `0x00000F00`

d) `0x00000000`

e) `0xFF94BEEF`

f) ∞

g) `1/3`

2   More Floating Point

As we saw above, not every number can be represented perfectly using floating point. For this question, we will only look at positive numbers.

2.1 What is the next smallest number larger than 2 that can be represented completely?

2.2 What is the next smallest number larger than 4 that can be represented completely?

2.3 What is the largest odd number that we can represent? Hint: at what power can we only represent even numbers?

3 RISC-V Instructions

3.1 Assume we have an array in memory that contains `int *arr = {1,2,3,4,5,6,0}`. Let register `s0` hold the address of the element at index 0 in `arr`. You may assume integers are four bytes and our values are word-aligned. What do the following snippets of RISC-V code do? Assume that all the instructions are run one after the other in the same context.

a) `lw t0, 12(s0)`

b) `sw t0, 16(s0)`

c)
```
slli t1, t0, 2
add t2, s0, t1
lw t3, 0(t2)
addi t3, t3, 1
sw t3, 0(t2)
```

d)
```
lw t0, 0(s0)
xori t0, t0, 0xFFF
addi t0, t0, 1
```

4 RISC-V Memory Access

Using the given instructions and the sample memory array, what will happen when the RISC-V code is executed? For load instructions (`lw`, `lb`, `lh`), write out what each register will store. For store instructions (`sw`, `sh`, `sb`), update the memory array accordingly. Recall that RISC-V is little-endian and byte address-able. For any unknown instructions, use the CS 61C reference card!

4.1

```
1  li t0 0x00FF0000
2  lw t1 0(t0)
3  addi t0 t0 4
4  lh t2 2(t0)
5  lw s0 0(t1)
6  lb s1 3(t2)
```

Address	Value
0xFFFFFFFF	
	...
0x00FF0004	0x000C561C
0x00FF0000	36
	...
0x00000036	0xFDFDFDFD
	...
0x00000024	0xDEADB33F
	...
0x0000000C	0xC5161C00
	...
0x00000000	

What value does each register hold after the code is executed?

t0:

t1:

t2:

s0:

s1:

4.2 Update the memory array with its new values after the code is executed. Assume each byte in the memory array is initialized to zero.

```
 1  li t0 0xABADCAF8
 2  li t1 0xF9120504
 3  li t2 0xBEEFDAB0
 4  sw t0 0(t1)
 5  addi t0 t0 4
 6  sh t1 2(t0)
 7  sh t2 0(t0)
 8  lw t3 0(t1)
 9  sb t1 1(t3)
10  sb t2 3(t3)
```

Address	Value
0xFFFFFFFF	0x00000000
	...
0xF9120504	
	...
0xBEEFDAB0	
	...
0xABADCAFC	
0xABADCAF8	
	...
0x00000000	0x00000000

5   Lost in Translation

5.1 Translate the code verbatim between C and RISC-V.

C	RISC-V

<pre>// s0 -> a
// s1 -> b
// s2 -> c
// s3 -> z
int a = 4, b = 5, c = 6;
int z = a + b + c + 10;</pre>	

<pre>// int *p = intArr;
// s0 -> p;
// s1 -> a;
*p = 0;
int a = 2;
p[1] = p[a] = a;</pre>	

<pre>// s0 -> a,
// s1 -> b
int a = 5;
int b = 10;
if (a + a == b) {
  a = 0;
} else {
  b = a - 1;
}</pre>	

<pre>// Compute s1 = 2^30
int s0 = 0;
int s1 = 1;
for (; s0 != 30; s0 += 1) {
  s1 *= 2;
}</pre>	

<pre>// s0 -> n
// s1 -> sum
for (int sum = 0; n > 0; n--) {
  sum += n;
}</pre>	