f) In our 32-bit single-precision floating point representation, we decide to convert one significand bit to an exponent bit. How many **denormalized numbers** do we have relative to before? (Circle one)

More Fewer

Rounded to the nearest power of 2, how many denorm numbers are there in our new format?
(Answer in IEC format)

______________

Questions (e)-(g) concern the *IEEE floating point standard*.

e) What `float` is encoded by the following bits: **`0xc14c0000`**? ____________________
(show all work here)

f) What is the smallest positive normalized number? Number: _______ encoded as `0x`____________
(show all work here)

g) Write the ~~MAL MIPS~~ x86-64 function **`IsNotInfinity`** to return non-0 if the input is **NOT** $\pm\infty$, 0 if it *is* $\pm\infty$.

```
IsNotInfinity:  movl    %edi, %eax

                _____   $1, %eax      # make +/- Inf look the same

                _____   ______________________________

                ret
```

## **M3) *What is that Funky Smell? Oh, it's just Potpourri…*** (10 pts, 20 mins)

a) How many non-negative `float`s are < 2 ? ___________ *(you must show your work above for credit)*

## **Question 4**: *Let Me Float This Idea By You* (9 Points, 16 Minutes)

For a very simple household appliance like a thermostat, a more minimalistic microprocessor is desired to reduce power consumption and hardware costs. We have selected a **16-bit** microprocessor that does not have a floating-point unit, so there is no native support for floating point operations (no `float`/`double`). However, we'd still like to represent decimals for our temperature reading so we're going to implement floating point operations in software (in C).

a) Define a new variable type called `fp`:

__________________________________________

We have decided to use a representation with a **5-bit exponent field** while following all of the representation conventions from the MIPS 32-bit floating point numbers **except denorms**.

Fill in the following functions. Not all blanks need to be used. <u>You can call these functions and assume proper behavior regardless of your implementation</u>. Assume our hardware implements the C operator ">>" as *shift right arithmetic*.

b)

```
/* returns -num */
fp negateFP(fp num) {
      return ______________________________________________________;
}
```

c)

```
/* returns the signed value of the exponent */
int getExp(fp num) {
      _____________________________________________________________

      return ______________________________________________________;
}
```

d)

```
/* multiplies floating point num by 2^n, while detecting over/underflow */
/* remember, there are no denorms */
fp multPow2(fp num,int n) {
      _____________________________________________________________

      if(___________________________________________) exit(1);  #overflow
      if(___________________________________________) exit(-1); #underflow
      _____________________________________________________________

      return ______________________________________________________;
}
```

SID: ________________

## Question 5: Floating Point (10 pts)

Assume integers and IEEE 754 single precision floating point are **32 bits wide**.

a) Convert from IEEE 754 to decimal: **0xC0900000**

b) What is the smallest *positive* integer that is a power of 2 that can be represented in IEEE 754 but not as a signed int? You may leave your answer as a power of 2.

c) What is the *smallest positive* integer `x` such that `x + 0.25` can't be represented? You may leave your answer as a power of 2.

d) We have the following word of data: **0xFFC00000**. Circle the number representation below that results in the *most negative number*.

Unsigned Integer    Two's Complement    Floating Point

e) If we decide to stray away from IEEE 754 format by making our Exponent field 10 bits wide and our Mantissa field 21 bits wide. This gives us (circle one):

MORE PRECISION // LESS PRECISION

SID: ________________

## **Question 1: Number Representation** (8 pts)

---

a) Convert `0x1A` into base 6. Don't forget to indicate what base your answer is in!

b) In IEEE 754 floating point, how many numbers can we represent in the interval [10,16)? You may leave your answer in powers of 2.

c) If we use 7 Exponent bits, a denorm exponent of -62, and 24 Mantissa bits in floating point, what is the largest positive power of 2 that we can multiply with 1 to get *underflow*?