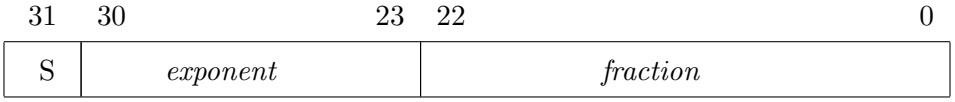
Question 1 (20 points): Write a subroutine called `BiggerThanTen` that receives a value x in `$a0`. The value x is represented in 32-bit IEEE 754 floating-point representation. `BiggerThanTen` returns one of the following combination of values:

Return Values		Meaning
$v1	**$v0**	
1	1	x is +infinity or -infinity
1	0	x is not a number
0	1	x is larger than +10
0	0	x is smaller than or equal to +10

Recall that the 32-bit IEEE 754 representation has the following specification:



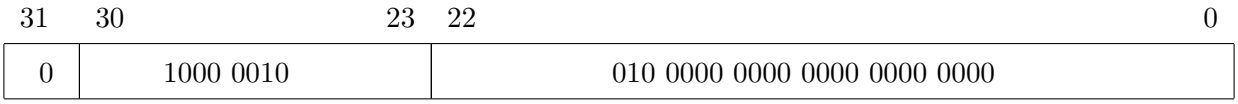
$$N = \begin{cases} (-1)^S \times 0.fraction \times 2^{-126} & \text{if } exponent = 0 \\ (-1)^S \times 1.fraction \times 2^{exponent-127} & \text{if } 0 < exponent < 254 \\ (-1)^S \times \infty & \text{if } exponent = 255 \text{ and } fraction = 0 \\ NaN & \text{if } exponent = 255 \text{ and } fraction \neq 0 \end{cases}$$

- **(10 points)** What is the binary representation of +10.0 in the IEEE 754 floating-point representation?

$$10 = 2^3 + 2^1 = 01010 = 1.01 \times 2^3$$

$$\text{exponent} - 127 = 3 \Rightarrow \text{exponent} = 130$$

Thus, the binary representation of +10 is:



- **(20 points)** Write the MIPS subroutine `BiggerThanTen`. Follow all the MIPS subroutine calling conventions. You are not allowed to use any floating point instructions in your subroutine.

```
BiggerThanTen:          sll   $t0, $a0, 1
                        srl   $t0, $t0, 24                  # $t0 ← exponent
                        li    $t1, 255
                        bne   $t0, $t1, Number              # if exponent ≠ 255
                        li    $v1, 1
                        move  $v0, $zero
                        sll   $t1, $a0, 9                   # $t1 ← fraction << 9
                        bne   $t1, $zero, NaN               # if fraction ≠ 0
                        li    $v0, 1                        # It is +/- infinity
NaN:                    jr    $ra                           # It is either +/- infinit
Number:                 move  $v1, $zero
                        move  $v0, $zero                    # Assume $a0 ≤ 10
                        srl   $t1, $a0, 31                  # $t1 ← sign
                        beq   $t1, $zero, positive          # if sign is positive
                        jr    $ra                           # $a0 is negative
positive:               li    $t1, 130
                        bge   $t0, $t1, exp_big_enough
                        jr    $ra                           # exponent is too small
exp_big_enough:         sll   $t2, $a0, 9
                        srl   $t2, $t2, 9                   # $t2 ← fraction
                        lui   $t3, 0x0020                   # $t3 ← 0x 0020 0000
                        bgt   $t2, $t3, fraction_big_enough
                        jr    $ra                           # $a0 ≤ 10
fraction_big_enough:    li    $v0, 1                        # $a0 > 10
                        jr    $ra
```