Laboratory 6: Prelab Exercise #3

Date ______________ Section ______________

Name ______________

Floating Point Arithmetic in MIPS

During this exercise you will use floating-point instructions. Since the SPIM simulator does implement only partially the IEEE-754 standard, this exercise will ignore most issues related to the standard, like ways to handle overflows, underflows, denormalized representation, rounding, etc. The focus will be on floating point number representation.

Floating point numbers

The IEEE-754 standard reserves several bit patterns to have special meaning. In other words not all bit patterns represent some number.

Special bit patterns in IEEE-754

Sign bit	Exponent	Significand	Comment
`x`	`0..0`	`0..0`	Zero
`x`	`0..0`	`not all zeros`	Denormalized number
`0`	`1..1`	`0..0`	Plus infinity (+inf)
`1`	`1..1`	`0..0`	Minus infinity (-inf)
`x`	`1..1`	`not all zeros`	Not a Number (NaN)

Infinity does not mean mathematically infinite, rather something too big to be represented. An overflow can return an +inf or a -inf (though the standard also provides a mechanism to determine the correct result in case of overflows). Some operations on infinity return yet another infinity as a result.

Table 1: Some operations with infinity

Operation	Result	Comment
`x + ( inf)`	`inf`	x finite
`x − (+inf)`	`−inf`	x finite
`(+inf) + (+inf)`	`+inf`	
`(−inf) + (−inf)`	`−inf`	
`x * (+inf)`	`+inf if x>0, −inf otherwise`	x nonzero

There can be a positive zero if the sign bit is 0 and a negative zero (the sign bit is 1). Denormalized numbers are included in IEEE-754 to handle cases of exponent underflow (very small numbers).

A *NaN* (sometimes denoted by *nan*) is used to represent an indeterminate result. There are two kinds of NaNs, signaling and quiet: the actual bit pattern in the significand field is used to differentiate between them, and it is implementation-dependent. A signaling NaN can be used for instance for uninitialized variables: attempting to operate on a signaling NaN can cause a trap. Note that any operation on a signaling NaN will have as a result a quiet NaN. Operating on a quiet NaN simply returns another NaN without generating any exception.

Table 2: Some operations which produce a quiet NaN

Operation	Comment
`x + (NaN)`	Any operation on a quiet NaN (addition in this example)
`(+inf) + (-inf)`	
`0 * ( inf)`	
`0/0`	
`inf/inf`	
`x%0`	The remainder of division by 0
$\sqrt{x}$, `x < 0`	

Step 1

Create a program named *lab6.9.asm* which:

- declares the variables *Zero.s*, *PlusInf.s*, *MinusInf.s*, *PlusNaN.s*, *MinusNaN*, of size word, initialized with the bit patterns corresponding to zero, plus infinity, minus infinity, positive Not a Number, and negative Not a Number respectively, in single precision representation
- declares the variables *Zero.d*, *PlusInf.d*, *MinusInf.d*, *NaN.d*, initialized with the bit patterns corresponding to zero, plus infinity, minus infinity, and Not a Number respectively, in double precision
- loads these variables in floating-point registers, starting with **`$f0`**
- prints, starting with **`$f0`**, the contents of those registers where variables have been loaded; print a newline (\n) character after each value.

Run the program. Fill out the following table

Printed output when running *lab6.9.asm*

Variable name	Printed output

Printed output when running *lab6.9.asm*

Variable name	Printed output

Step 2

Create the program *lab6.10.asm* which:

- declares the same single precision variables as *lab6.9.asm*
- declares a float called *MyNum*, initialized to the first two digits of your Social Security Number
- performs an operation which uses *MyNum* and which generates infinity; print the result. Choose some operation from Table 1
- performs an operation that generates NaN and print the result; choose some operation from Table 2

Run the program. Fill out the following table

Printed output when running *lab6.10.asm*

MyNum	Operation implemented	Printed result

Q 1:

What is the bit pattern for the largest possible single precision floating point number? Write it down in hexadecimal. .
