Quick Review
What is the instruction format for each of the following instructions?
```
add $s0, $s1, $s2                    addi $s0, $s1, 5
beq $zero, $zero, LABEL              slti $s0, $s1, 0
jr $ra                              j LABEL
```

Translate the following instruction into hexadecimal (the `sra funct` field is 3):
```
sra $8, $9, 16
```

Floating Point Number Representation
In general, floating point numbers are represented using a sign and magnitude model. As in integer sign and magnitude, a floating point number's sign is represented by the leading bit (1 for negative numbers, 0 for positive). The magnitude of the float is broken down into an exponent field and a significand or fraction field.

Sign	Magnitude	
Sign	Exponent	Significand

$$\text{float} = (-1)^{\text{sign}} \times (1.\text{Significand})_2 \times 2^{(\text{Exponent} - \text{Bias})}$$

This breakdown is much like standard scientific notation. The exponent determines the value of the bits in the significand (essentially defining an amount to shift the binary point from normalized form). The significand is similar to the mantissa in scientific notation.

Rounding Modes:
IEEE 754 defines 4 rounding modes to determine how the extra two guard bits are used:

Round Towards +∞	Round Towards -∞	Truncate	Unbiased
round "up"	round "down"	round towards 0	round to even

Rounding Exercises
Round the following binary numbers to the nearest integer using each of the four modes:
0.00   0.01   0.10   0.11   1.00   1.01   1.10   1.11

Single Precision Floating Point:
```
31  30    23 22                        0
 S  EEEEEEEE FFFFFFFFFFFFFFFFFFFFFFF
```
(with an exponent bias of 127)

Double Precision Floating Point:
```
63 62      52 51                       0
 S  EEEEEEEEEEE FFFFFFFFFF...FFFFFFFF
```
(with an exponent bias of 1023)

Exponent	Significand	Meaning
0	0	0
0	Non-zero	Denorm
1~254	Anything	Float
255	0	Infinity
255	Non-zero	NaN

Floating Point Exercises
Convert the following decimal numbers into binary (not float).
1.5   0.25   0.8   -16.5

Give the best hex representation of the following numbers (using single precision floats):
1.0   -7.5   (1.0/3.0)   (186.334/0.0)

What is the value of the following single precision floats?
0x0   0xff94beef   0x1

Disassembly
The process of translating raw binary instructions into MIPS is called disassembly. Given a simple program, it is possible to translate from a raw binary all the way back to an equivalent C program.

The first step in disassembling a single instruction is to figure out what instruction format it is. This is easy, because all instruction formats conveniently reserve the first 6 bits for the `opcode` field. From the `opcode`, the rest of the bits can be interpreted appropriately.

Disassembly Exercises
Be a processor! Translate the following hex instructions into MIPS:

```
0x8c880000
0x2108ffff
0xaca80000
0x03e00008
```

MAL vs. TAL
MIPS comes in two different flavors: MAL and TAL. MIPS assembly language (MAL) is the more programmer (or lazy compiler) oriented version. It abstracts away the details of immediate field limitations and extends the instruction set. True assembly language (TAL) is the stricter, processor friendly MIPS. There is a one-to-one translation from TAL instructions to binary executables. It is the job of the assembler to translate from MAL to TAL. A single MAL pseudoinstruction might become several TAL instructions.

MAL vs. TAL Exercises
Be an assembler! Translate the following MAL program to TAL:

```
Foo:  bge $s0, $s1, Bar
      swap $s0, $s1
Bar:  beqi $s0, 100, End
      incr $s0
      j Bar
End:  add $s0, $s0, -100
```