Logical Instructions in MIPS

In MIPS (like in C) → 1 = true, 0 = false

e.g.

```
and $t0, $t1, $t2        # bitwise and
```

Example:

$t1	1	1	1	0	1	0	0	0
$t2	1	1	1	1	1	1	1	1
$t0	1	1	1	0	1	0	0	0

`or and xor` are similar:

	and		**or**		**xor**	
	0	1	0	1	0	1
0	*0*	*0*	*0*	*1*	*0*	*1*
1	*0*	*1*	*1*	*1*	*1*	*0*

Just like arithmetic instructions `andi, ori`, `xori` are the same except the third operand is an immediate instead of a register.

Shift

```
sll $t2, $t1, 1    # Shifts left logical
```

Example:

$t1	1	1	1	0	1	0	0	0
$t2	1	1	0	1	0	0	0	0

When we shift left logically we add zeros to the right and the leftmost bit drops off the edge.

`sllv` (shift left logical variable) is the same except the last operand is a register (shift amount) instead of an immediate.
Shift Amount is the 5 least significant bits of the register as an unsigned integer.

`srl and srlv` (shift right logical) is the same concept.

`sra` (shift right arithmetic) is like `srl` except the left is filled with the sign extension instead of zero. There also exists a `srav`.

Example:

$t1	1	1	1	0	1	0	0	0
$t2	1	1	1	1	0	1	0	0

Note: To multiply by 2^n simply shift n bits to the left. To divide shift n bits to the right.

<u>Practice:</u> Write a method that receives a bit pattern in `$a0` and:

1. Returns in `$v0` 0 and 1 depending on the most significant bit of `$a0`:

Solution:

```
srl   $v0, $a0, 31
```

2. Returns in `$v0` 0 and 1 depending on the least significant bit of `$a0`:

Solution:

```
andi  $v0, $a0, 1        # '1' functions as a "mask"
```

3. Returns in `$v0` 0 and 1 depending on bit `$a1` of `$a0`:

Solution:

```
srlv  $v0, $a0, $a1
andi  $v0, $v0, 1
```

4. Set bit # 10:

Solution:

```
ori   $v0, $a0, 1024     # 1024 = 2^10
```

5. Flip (Replace 0's and 1's):

Solution:

```
xor   $v0, $a0, -1
```

6. Clear bit # 10:

Solution:

```
addi  $v0, $0, 1024      # 2^10 = 1024
xor   $v0, $v0, -1       # flip 1024
and   $v0, $a0, $v0
```