CS061 – Lab 06
Advanced Bit Manipulation

1 High Level Description

The purpose of this lab is to give you some experience in doing some advanced bit manipulation techniques, making use of both left- and right-shifting.

2 Our Objectives for This Week

1. Exercises 01 – Review-ish
2. Exercise 02 – Counting bits
3. Exercise 03 – Right Shift

REMEMBER: ALL your programs must now be written as subroutines!!

Call your subroutines with the JSRR instruction (since it has no PCOffset-style limitations on how far it can jump ☺)

And remember to include all four steps and be *especially* careful to properly follow **step 1** with regard to R7 (or your program won't work and you won't be able to figure out why!)

Exercise 01

Hey, remember that Programming Assignment you just did? Sweet, huh? Let's start out with that:

Specs:

1. Input any positive decimal number less than or equal to "32767" from the keyboard
2. Convert it into its numerical equivalent and put it into a register (you can use your Programming Assignment 04 code for this)
3. Add 1 to the number
4. Print the new value out to the console as a decimal number (i.e. the inverse of Programming Assignment 04). This step should be in its own subroutine.

Exercise 02

Write a subroutine counts the number of binary 1's in the number stored in a given register.

Specs:

1. The main code block (test harness) should ask the user to input a single character at the keyboard
2. The input should be "passed" as a parameter to the subroutine (i.e. the input character is put into a register that is assumed by the subroutine).
3. The subroutine should "return" the number of binary 1's in the input character in another register (i.e. it places the number of 1's in a register and doesn't restore that register before returning).
4. The main code block should then print the result in a reasonably intelligent format:
   Example: The ASCII code for a semi-colon (';') is #59 == x3B == b0000 0000 0011 1011
   i.e. the code for ';' contains five binary 1's, so your program will output something like:
   "The number of 1's is: **5**" or "I love my T.A. * 10^**5**".

Exercise 03

Ok, here we go. Are you feeling advanced? You *look* advanced. You're awesome! You can totally do this! Ready? Ok.

Build a subroutine that takes, as a parameter, the value of a register and **right-shifts** it by one bit.

Example:

```
(R1) ← x1234     ; (b0001 0010 0011 0100)
[call the right-shift subroutine]
[now (R1) == x091A == b0000 1001 0001 1010]
```

Note how we shifted-in a 0 when we did the right-shift (see the red thingy? ☺)

Discussion:

Alright, let's think about this thing. When we **left-shift**, we are doubling the number (aka: multiplying it by two) (aka: adding it to itself) (aka: ADD R1, R1, R1), right? So if left-shift is multiplication, then what might **right-shift** be?

Yep, you guessed it: Division. If you left-shift 2, you get 4. If you right-shift 4, you go back to having 2. So, division. Right. Got it.

"How in blazes are we suppose to divide?!" the TA said rhetorically.

[Whacky Student]: If left-shifting is ADDing a number to itself, then right-shift must be subtracting it from itself, right?

[TA]: *blank stare*

[Whacky Student]: …

[TA]: *blank stare*

[Less Whacky Student]: *whispers* "Dude, that would make it zero."

[Whacky Student]: *facepalm*

Well, sadly, there is no super-simple way to perform a binary right-shift. Still, the way to do it isn't insanely difficult. Let's think about it:

- Left-shifting is short and sweet: Add the number to itself.
- Right-shifting is not quite as simple. There is no way to directly do it.
- Hmm… how else can we mess around with the bits? We can shove them left and shift-in a 0 every time (that's left-shifting)… what if, instead of always shifting in a 0 on the right-hand side, we just **rotated** the bits (i.e. take the MSB and stuff it on the right-hand side of the bit pattern)?

Hmmm:

- Given the 4-bit (unsigned) number xE == b1110 == #14
- [Rotate left]
- Now we have b1101 == #13
- [Rotate left]
- Now we have b1011 == #11
- [Rotate left]
- Now we have b0111 == #7
- Set the MSB to 0
- Ok, this is boring. Let's stop.

Hey wait! Oh my gosh!
We have 14/2 == 7 now!
How'd that happen?!?
[This is where you, the Less Whacky Student, fill in the blanks ☺]

Note: This only works for **positive** values. For 2 points of extra credit, see if you can make it work for negative values as well!