Introduction to Numbers

Exercise 0: Write a base-10 to binary converter.

Exercise 1: Write a binary to base-10 converter.

Exercise 2: Write a binary to hexadecimal converter by converting the number to base-10 first then to hexadecimal.

Exercise 3: Write a binary to hexadecimal converter *without* converting the number first limited only to the following operations: (left shift, right shift and "ands (&)").

Exercise 4: Write a program that adds two base-10 numbers together *without using mathematical operators*.

Exercise 5: Write a program that subtracts two base-10 numbers together without using mathematical operators

Exercise 6: Write a program that adds, multiplies, and subtracts two binary digits *all steps must be shown for multiplication in other words no "*" for multiplication*.

Exercise 7: Write a program that given two base-10 numbers will spit out the: and, or, xor, and negation of them.

Binary and Programming

Exercise 0: Given a 4 bit number, using a sign bit, give the maximum and minimal number.

Exercise 1: Write a program that when given a 4-bit number will spit out the corresponding base-10 number using the last bit as a sign number. All input that’s bigger than 4-bits must be discarded and error given.

Exercise 2: Print out the decimal value for letters A-Z, a-z, and 0-9 in Python.

Exercise 3: Convert the following irrational number to the Floating Point Format: -1.25. *All steps must be given in order* [**No exceptions**]. Note the auto-grader will check for “Step 1, Step 2, Step 3…” and if those are not there it is an automatic fail.

Introduction to Assembly

Exercise 0: Change the number for the exit code from our very first project and see what happens.

Exercise 1: Make an infinite loop that continually adds 2 to a register of your choosing

Exercise 3: Translate the following Python Code into Assembly [does not need to compile but MUST function]:

```
y = 0
for x in range(5, -5, -1):
    y += x
```

Exercise 4: Translate the following Python Code into Assembly that compiles, functions and exits with the value of the resulting operations performed:

```
y = 5
for x in range(0, 10, 2):
        y <<= 1

y -= 123
if y % 2 == 0: #Hint: Look at idiv
        y <<= 1
elif y +5 <= 42:
        y >>= 1
elif y + 10 > 42:
        y += 1
else:
        y = 3
```

Functions in Assembly

Exercise 0: Modify max.s so it finds the minimum instead.

Exercise 1: Using the C calling convention implement the following as a *function* that will be *called* in _start:

$$f(x, y, z) = \begin{cases} x * y \; if \; z > 0 \\ x + y \; if \; z < 0 \\ x - y \; if \; z = 0 \end{cases}$$

Must *restore the stack* and remove all local variables if any are used before exiting the program.

Recursion in Assembly

Exercise 0: Implement the following recursively (checks if an input number is even or odd):

$$f(x) = \begin{cases} 1\ if\ x\ is\ even \\ 0\ if\ x\ is\ odd \end{cases}$$

Hint it will look like the following:

$$f(x) = \begin{cases} 1\ if\ x\ is\ 0 \\ 0\ if\ x\ is\ 1 \\ f(x-2)\ all\ other \end{cases}$$

Exercise 1: Implement factorial recursively. Here is the recurrence formula for reference:

$$x! = \begin{cases} 1\ if\ x = 0 \\ (x-1)! * x\ if\ x > 0 \end{cases}$$