CS Basics - Exercises
Bits and Branching

E. Benoist P. Mainini

Fall Term 2022-23

1 Exponentiation

Create a program that computes the power 2^{64}. Store the values as `x` and `exponent` in memory. Then, write the code to compute the value $x^{exponent}$. Place the result on the stack in little endian order. Examine it using `gdb` – is it correct?

2 Input and Output

1. Create a program that reads up to 16 characters from `stdin`, and then outputs the same characters back on `stdout`.
2. Create a program similar to the first, but reading characters endlessly until `stdin` is closed (e.g. `Ctrl-d` is pressed or the end of the file has been reached).
3. Create a modified version of the previous program, which converts upper case letters (`"A"` – `"Z"`) to lower case. Ensure that only letters are converted and all other characters are output without modification!

3 Compute Binary Logarithms

The goal of this exercise is to write a program, which calculates the *binary* logarithm of the number 0x20. For this, divide 0x20 by 2 as long as the result is not 0. The binary logarithm is then the number of divisions required.

Your program should return the binary logarithm as the *return code*, which can then be inspected in the shell as follows:

```
$ ./exercise3
$ echo $?
```

4 Convert Numbers from/to ASCII

In this exercise, an ASCII representation of a decimal number should be read from `stdin` and converted to a number which can be used for performing calculations (i.e. in a register). The biggest number to be entered is $2^{64} - 1$.

After reading and converting the number, its binary logarithm should be calculated as in the previous exercise. The result should then be returned as a decimal number on `stdout`.

Additionally, the program should return with a return code $\neq 0$ on error, 0 otherwise.

It might be easier to break the task into smaller steps (maybe creating different programs for each):

1. Do not care about reading the number from `stdin` first, simply define it as an ASCII string in `.data` and implement the conversion. Store the result in a register and check it with the debugger.

2. Start by converting a one-digit number. When you have understood the principle, increase the number of digits.

3. When number conversion works, implement reading it from `stdin`.

4. Calculate the logarithm as in the previous exercise.

5. Convert the result back to ASCII. The conversion is almost the same as when converting the digits from ASCII.
   Hint: The result has no more than two digits, use division (`div`) to separate them.

4.1 Example Output

Below are some examples of logarithms calculated with the program (first number is the input, second number the logarithm):

```
$ ./exercise4
0
00

$ ./exercise4
1
00

$ ./exercise4
2
01
```

```
$ ./exercise4
32
05

$ ./exercise4
18446744073709551615
63

$ ./exercise4
18446744073709551616
00
```