CCPS310 Lab 3 – ARC III

Preamble

In this lab we tie everything together. You'll write several subroutines, parse input from the keyboard and write output to the console.

Lab Description

For this lab you will write a single assembly program that acts as a simple arithmetic parser and calculator. Your program will begin by reading in three characters total. First, a positive, single-digit integer [0-9] (yes, operands can be zero!). Second, one of the following symbolic operators: **+**, **-**, *****, **/**, or **^**. Third, another positive, single-digit integer [0-9]. If any of these three inputs is invalid (doesn't match the above criteria), your program should print "Invalid input!" to console and then halt. Assuming the three inputs are valid, you will handle them according to which symbolic operator was chosen:

Addition **(+)** and Subtraction **(-)**

Addition and subtraction can be performed inline. Simply add or subtract the two operands (after converting them to integers of course!). You are not required to write separate subroutines for performing these two.

Multiplication **(*)**, Integer Division **(/)**, and Exponentiation **(^)**

We saw how to implement multiplication in class, and you implemented integer division and exponentiation in lab 2. It's your job to implement each of these in their own separate subroutines. Your subroutines should use a shared memory area to pass the arguments and return the result. You may use one single shared memory area for all three. That is, you don't need three separate regions, one for each subroutine. Your subroutines should be labelled **`mult:`**, **`div:`**, and **`exp:`** respectively.

Once you have parsed the inputs and computed the result, you will call another subroutine to print the full equation and the result. To make things simpler, you may print numeric values in hexadecimal. This makes converting the result to characters much easier. Below are several input/output examples:

Input: 4, *, 5	**Output:** 0x4 * 0x5 = 0x14
Input: 9, ^, 9	**Output:** 0x9 ^ 0x9 = 0x17179149
Input: a, b, c	**Output:** Invalid input!

This output subroutine should be labelled **`print:`**, and will use a separate shared data area for passing arguments. That is, **`print:`** may not use the same shared data area as the subroutines for multiplication, division, and exponentiation. Additionally, your print subroutine should correctly ensure the console is ready before storing each character (as we saw in class).

Things to Consider

- Operands can be zero, but you may assume I will **never** attempt to ***divide*** by zero. Thus, you don't need any special logic for handling this. Every other operation should handle zero correctly, however. For example: 0/x = 0, x^0 = 1, and so on.
- There are no restrictions on how many registers you use, how you label your program (other than the subroutine labels described previously), or where you load things in memory.
- Remember that characters are read as ASCII values. The *character* '1' is not the same thing as the *integer* 1! Operands (numbers) must be converted. Operator symbols can just be used directly in your selection logic.
- One of the trickier aspects of this assignment is going to be converting large integer results to their respective sequence of ASCII characters. Remember that every string of 4 binary bits can be directly converted to a hex digit. This hex digit can then be converted to it's corresponding ASCII character.
- It's OK if your printed results contain leading zeros. For example, if you print 4 as **0x00000004**, that's fine.
- As is often the case, this assignment is daunting when viewed from the top down. Break it into parts, implement the easy stuff first. We saw sample code for most of what you have to do.

Marking

This lab is out of **20 marks**. The breakdown is as follows:

(2 marks) Correctly read input and converted operands from characters to integers.

(1 mark) Addition and subtraction works properly. These are easy.

(6 marks) Multiplication, division, and exponentiation subroutines function properly and produce the correct results.

(3 marks) Shared data area for passing arguments and returning result is correctly implemented and utilized for the above subroutines.

(4 marks) Output subroutine (**`print:`**) is correctly implemented and displays *something*. It uses its own shared data area for receiving data.

(4 marks) Full equation and result correctly printed to console. Operands and result should be their numerically correct values.

Submission

Labs are to be submitted ***individually***! Submit your Lab3.asm file on D2L under Lab #3.