CS220
Introduction to Computer Organisation
Lab 6

Amey Karkare

This is the second and last lab on assembly programming. You will be implementing MIPS assembly programs for given problems and run them on SPIM simulator[1]. Refer to MIPS manual for instruction set, and SPIM manual for running the programs.

NOTE:

1. For each problem, **20%** marks are reserved for **useful** comments. Comments like

   ```
   add  $11, $12, $13   # adding $12 to $13, storing in $11
   ```

   are useless, and should be avoided.

2. Usage conventions for MIPS registers have to be honored.

3. For each task, the function to be implemented has to be free of input/output calls. Parameter should be passed as argument, and result should be returned as return value. Error reporting using output call is ok, but should not clutter your main logic.

4. For each task, you have to additionally implement a `main` function that reads input and produces output. This is *not* mentioned in the individual tasks, but you have to do it.

5. You are encouraged to use helper functions to create a modular design.

6. Assume the inputs to be non-negative (≥ 0) for each task, so there is no need to test for negative numbers.

[1] Since SPIM simulator supports almost all of the MIPS instruction set, the programs you write need not be restricted to SMIPS instruction set.

1 Run-length encoding [10]

In this task, you implement the function `runlength` in `runlength.asm` for *run-length encoding* data compression. In this method, the consecutive repetitions of elements are encoded as terms `N E` where `N` is the number of repetitions of the element `E`.

The input will be positive integers (> 0), separated by space. The input is terminated by a 0. You have to store the input (without the terminating 0) as a linked list. The function, `runlength`, will take this linked list as argument and generate run-length encoding in another linked list.

The main program will print the result as space separated numbers.

1.1 Examples

Input: 1 1 5 5 5 5 10 12 12 12 12 12 12 12 0
Output: 2 1 4 5 1 10 7 12

Input: 1 2 3 4 5 0
Output: 1 1 1 2 1 3 1 4 1 5

2 Multiplying Big Numbers [20]

You have to write program to multiply BIG numbers (of arbitrary sizes). Both the multiplier and multiplicand should be inputs from the user. You need to store each digit of the number on a linked list, and produce the result as another linked list.

Your function, `multiply`, will take the two numbers as linked list and produce the product as another linked list.

Multiplier and multiplicand are entered by user as individual digits separated by space. A number greater than 9 in input signifies end of the operand. So, 12345 could be represented as: 1 2 3 4 5 11, where 11 (> 9) signifies end of the number.

2.1 Examples

These examples are for illustration purpose only. Real tests will use really big numbers.
Input: 1 2 3 4 5 6 7 8 9 10 1 0 10
Output: 1 2 3 4 5 6 7 8 9 0

Input: 1 2 3 4 5 6 7 8 9 11 9 8 7 6 5 4 3 2 1 12
Output: 1 2 1 9 3 2 6 3 1 1 1 2 6 3 5 2 6 9