Assembly and Machine Language Homework 4

Hamid Mohammadi

Your task is to write a standalone assembly program (without using driver.c) that reads two integers A and B from the standard input, calculates A^B, and prints the result on the standard output. In this homework, you are not allowed to use the IO functions from the book or the C libraries and you must implement the required functions yourself. These functions are as follows:

1. **read_int**: Write an assembly function which reads a **signed** integer from the standard input and stores it in **EAX**. You are only allowed to use the 32-bit Linux system call **sys_read**. You cannot use the C libraries. The input may (or may not) start with a sign character (a '-' or '+').
2. **pow_int**: Write an assembly function which calculates A^B, where A and B are the arguments passed to the function using the stack. Return the result in the **eax** register. The first argument A can be positive, negative or zero, while B is always nonzero.
3. **print_int**: Write an assembly function that prints the content of the eax register as an integer on the standard input. You can only use the Linux system call **sys_write**. Please keep in mind that the result can be negative, thus, you must print a preceding negative sign '-' for negative numbers.

In case you need help with system calls, refer to this website:
https://www.cs.utexas.edu/~bismith/test/syscalls/syscalls.html

You must write a standalone assembly program and you must provide a **Makefile** for your program. Please organize your program in these 4 files:

1. **Makefile:** To build your program
2. **main.asm**: Contains **_start** and **pow_int** functions
3. **io.asm**: Contains **read_int** and **print_int** functions
4. **io.inc**: The header file for the **io.asm** to be Included in main.asm. Reading asm_io.inc gives you an idea about what to write in **io.inc**.

Extra credit

Write the above as a **64-bit** application. You must use 64-bit specific operations and registers (use RAX, RBX, RSP, RBP instead of EAX, EBX, ESP, EBP and so forth). Remember that

- in the 64-bit mode, the addresses are **8 bytes** wide. This includes the return address pushed on the stack for function calls.
- The PUSH and POP instructions put/remove 8-byte blocks on/from the stack.
- The return values must get stored in RAX.

- Your program must accept 8-byte signed integers. All calculations must be performed in 64-bit registers/memory units. The output might be 64 bits wide.
- You are allowed to use R8 - R15 registers.
- The 64-bit system calls differ from the 32-bit ones, look here http://blog.rchapman.org/posts/Linux_System_Call_Table_for_x86_64

Sample inputs and outputs:

inputs:	*output:*
-3 4	81

inputs:	*output:*
-4 3	-64

Sample 64 bits inputs and outputs:

inputs:	*output:*
1073741824 2	1152921504606846976

inputs:	*output:*
-6 21	-21936950640377856