Floating Point Operations

TABLE OF CONTENTS

Pintos currently does not support floating point operations. You must implement such functionality so that both user programs and the kernel can use floating point instructions.

Implementation details

This may seem like a daunting task, but rest assurred, the operating system doesn't have to do much when it comes to implementing floating point instructions. The compiler does the hard work of generating floating point instructions, while the hardware does the hard work of executing floating point instructions. However, because there is only one floating-point unit (FPU) on the CPU, meaning all threads must share it. As a result, the operating system must save and restore floating-point registers on the stack during context switches, interrupts, and syscalls like it does for general purpose registers (GPR). However, contrary to GPRs, The FPU registers need to be initialized correctly when a new thread or process is created as well, which is different from the GPRs. Additionally, the operating system needs to initialize the FPU during startup. See Floating Point for more details.

The learning goal for this class is not for you to learn the ins and outs of floating point operations but rather understand the details of thread/context switching. If you find yourself reading through specialized floating operations dealing with the actual arithmetic itself, take a step back and reread through the task and Floating Point appendix.

Since this task deals with some of the OS initialization and context switching code, an error in these files may result in some gnarly and indecipherable errors that result from corrupting a thread. **We recommend you work on floating point operations after all the other tasks**.

Syscall signatures

As a chance for you to see floating point operations in practice and for us to test your code, you will also need to implement the `compute_e` syscall. The signature of the syscall is included below:

compute_e

```
double compute_e (int n)
```

This is similar to the `practice` syscall in that it's a "fake" system call that doesn't exist in any modern OS. This system call computes an approximation of e as $\approx \sum_{i=0}^{n} \frac{1}{n!}$ using the Taylor series for the constant e. Most of the mathematical logic for implementing this system call has been done for you in `sys_sum_to_e` in `lib/float.c`. Once you validate the argument `int n` from the system call, you can simply store the result of `sys_sum_to_e` in the `eax` register for return. This system call is not intended to be difficult to implement correctly.
