Find the faulting instruction

First, run `make` and `make check` in the `proj-pregame/src/userprog` directory, and observe that currently no tests pass.

For this assignment, we will step through the execution of the `do-nothing` test in GDB to learn how we can modify Pintos so that the test passes, and understand how Pintos's existing support for user programs is implemented.

`do-nothing` is the simplest test of Pintos's user program support. Take a look at `proj-pregame/src/tests/userprog/do-nothing.c`; it is a Pintos user application that does nothing. The single `return 162` statement in the main function returns the exit code `162` to the operating system. The specific value of the exit code is immaterial to the test; we chose a value other than 0 so that it's easier to track how the Pintos kernel handles this value through GDB (note `162 = 0xa2`).

When you ran `make`, `do-nothing.c` was compiled to create an executable program `do-nothing`, which you can find at `proj-pregame/src/userprog/build/tests/userprog/do-nothing`. The `do-nothing` test simply runs the `do-nothing` executable in Pintos.

View the file `proj-pregame/src/userprog/build/tests/userprog/do-nothing.result`. (Alternatively, you may also run `pintos-test do-nothing` in the terminal.) This file shows the output of the Pintos testing framework when running the `do-nothing` test. The testing framework expected Pintos to output `do-nothing: exit(162)`. This is the standard message that Pintos prints when a process exits (you'll encounter this again in Project Userprog). However, as shown in the diff, Pintos did not output this message; instead, the `do-nothing` program crashed in userspace due to a memory access violation (a segmentation fault). Based on the output of the `do-nothing` test, please answer the following questions on Gradescope:

1. What virtual address did the program try to access from userspace that caused it to crash? Why is the program not allowed to access this memory address at this point?
2. What is the virtual address of the instruction that resulted in the crash?
3. To investigate, disassemble the `do-nothing` binary using `i386-objdump` (you used this tool in Homework 0). What is the name of the function the program was in when it crashed? Copy the

disassembled code for that function onto Gradescope, and identify the instruction at which the program crashed.

4. Find the C code for the function you identified above (*Hint: it was executed in userspace, so it's either in* `do-nothing.c` *or one of the files in* `proj-pregame/src/lib` *or* `proj-pregame/src/lib/user`), and copy it onto Gradescope. For each instruction in the disassembled function in #3, explain in a few words why it's necessary and/or what it's trying to do. *Hint: read about 80x86 calling convention*.

5. Why did the instruction you identified in #3 try to access memory at the virtual address you identified in #1? Please provide a high-level explanation, rather than simply mentioning register values.
