PROGRAM 5

Description

The goal of this assignment is to support exception handling for C programs on the Intel 64 architecture.

The public interface to the exception handling mechanism is the following three functions:

- **int catchException(void)**
- **void cancelCatchException(void)**
- **void throwException(int exceptionNumber)**

catchException enables an exception handler for the current function. This function returns 0 on its initial return but will "re-awake" and return the number of the exception if an exception is thrown while the handler is active. Any exception handler active prior to the call to catchException will be temporarily disabled while the new handler is active and automatically re-enabled when the new handler is cancelled.

cancelCatchException cancels the currently active exception handler. If a handler was made inactive when the handler being cancelled was enabled, then the handler is now made active again.

In essence, handlers are stacked. A newly enabled handler is pushed on a stack of handlers and the topmost handler is always utilized when an exception is thrown. When a handler is cancelled it is popped from the stack,

If there is no active handler when **cancelCatchException** is called, then no action is taken. In essence the request to cancel is just ignored.

If an exception handler is still active after the function in which it was enabled returns, then the behavior is undefined. It is the user's responsibility to call **cancelCatchException** before the function returns. If this responsibility is not met, then strange things can happen.

throwException throws the given exception, which will be caught by the most recently enabled handler, if there is one. If there is no enabled handler, then the program is terminated with an appropriate message to **stderr** that gives the exception number. The standard C library function **exit** is used to terminate the function and the exception number is passed to **exit** as its argument.

It is a fatal error to pass 0 to **throwException**. An appropriate error message is printed to **stderr** and the program is terminated by passing **-1** to **exit**.

A thrown exception causes the cancellation of the handler that catches it.

The values of the callee-saved registers (rbx, r12-r15, rsp and rbp) should be restored at the time of the re-awakening of **catchException** to what they were at the call to **catchException**.

You will need to implement pieces of this assignment in Intel 64 assembly language. You should aim to minimize the amount of assembly language you need to write. You are not allowed to use the GNU C inline assembly language mechanism. I want you to put your assembly language code in a separate file.

You are not allowed to use the standard C library functions setjmp/longjmp or setcontext/getcontext. This assignment, in part, explores how those functions are implemented, so you need to implement them yourself.

All C code for this assignment should be placed in the file **eh.c** and all assembly language code should be placed in the file **asm.s**.

The three expected error messages from your eh.c module at the appropriate places should be:

```
fprintf(stderr, "malloc returned NULL in catchException!\n");
fprintf(stderr, "throwException: zero is not a valid exception number\n");
fprintf(stderr, "unhandled exception %d\n", exceptionNumber);
```

If any of your test cases fail because of a different error message we will credit your score as long as the error is being reported.

Note that you will need to correctly define all 3 of the above functions as per specifications. Faking it by invalidating the stack and just returning to the caller, having empty functions and returning, etc. will result in a score of 0 for the program (and the corresponding lab). Program 5 also expects dynamic memory allocation to keep track of the stack of frames, some tests may fail if you haven't done this correctly.