Lab 3:
Precise Trap

NAME: YUHAO ZHANG
STUDENT NUMBER: 2021533141
EMAIL: ZHANGYH7@SHANGHAITECH.EDU.CN

1 INTRODUCTION

"Precise trap" refers to a feature where the processor can report an exception or error condition at a well-defined instruction in the program's execution.

In this lab, we add a precise trap handling mechanism by implementing a system call to the original simulator. Below are the tasks in this lab.

- Write kernel code.
- Implement trap handler.

2 KERNEL CODE

The kernel code maintains an contiguous array and finds out the largest integer in it. For simplicity, we still write it with C language at first, and translate it into riscv binary code using the tool given in lab 0. In the kernel code I create an array with 20 elements in it, and use a for loop to find the largest element.

Because the kernel code do not use the data in the original code, so we can just treat them separately.

3 TRAP HANDLER

The entry of the trap handler is in the *handleSystemCall* function, by passing parameter 7 to it. When the exception occurs, the function *syscall* is called, the stage of the data path is stored, and the kernel code begins to run. When the kernel code finished, the program continues to run.

For simplicity, we use another simulator to represent the trap handler. It would not affect the state of the original code, which ensure the correctness of the original code. Because the stack address starts at 0x80000000 and goes down, we can use the address space 0x80000000-0xFFFFFFFF to store our kernel code.

4 OUT OF ORDER

Because of my implementation method, the precise trap mechanism implementation in out of order machine is not so different from the pipelined machine: when an exception occurs, the kernel code will be executed in another simulator, which maintains the original state of the simulator.

5 TEST

In my test case is designed base on the *test_arithmetic*, which I add a trap trigger to it.

To prove the correctness of my implementation, I added some flag in the test case, which marks where the kernel code begins and ends. You can see that when you run the test case, and you can also view it in the *test/test_trap.c*.

You can test it with the command below:

- ./Simulator ../riscv-elf/test_trap.riscv ../riscv-elf/kernel.riscv

Or you can get more information in the README.md.