CSC258H Week 12 Lab: Exceptions

1 Introduction

Last week, you implemented Larus's function call conventions in a recursive function. This week, we'll be working with a different control structure: exceptions (or interrupts, if you prefer). An exception is a hardware mechanism for executing code in a priveleged (system, not user) mode that allows anything in the computer's register set or memory to be changed. This mechanism is used to handle unexpected errors (faults) and also to invoke the operating system (system calls).

The heart of an exception is a *signal*. The signal stops the processor. Instead of continuing to execute the user program, the processor executes a special "exception handler" (or "trap handler") code that invokes exception handlers (often terminating the user program) or loads the operating sytem. **Before lab, review section 7 of Larus's assembly guide for implementation details.**

2 Encountering Exceptions

We will focus on the case where an exception occurs because of an error. Download a buggy assembly program from the course website. `buggy.s` contains a modified version of the "average" program that we used several weeks ago. After reviewing the code, assemble and execute the program in MARS. You'll see that it fails with a runtime exception (address out of range). These errors occur because the programmer assumed that the data segment begins at address 0.

Next, download the commented trap handler from the course website. Open the file in a text editor and scan the code to make sure that it corresponds with your understanding of exception handling in MIPS. Pay particular attention to the documentation at the top of the file which describes interrupt handling in MIPS and summarizes the code. Then, set MARS to use the trap handler that you downloaded. To do so, go to the *Settings* menu and select *Exception Handler...*, click the checkbox to include the exception handler in all future assemble operations, and browse to select the exception handler you just downloaded.

Run the buggy assembly program again. Instead of the runtime exception from before, you will see a series of "Exception 4" messages that "occurred and [were] ignored". Re-initialize the machine and set a breakpoint at the first instruction of the trap handler. (Note that the trap handler has been placed at 0x80000180 – an address far greater than the normal "code" segment!) Re-execute the program and single-step through the exception handling code, so you understand why these messages are being emitted.

3 Handling Exceptions

Your assignment is to modify the trap handler so that it handles the exceptions the program throws to allow it to produce the correct output. To do so, you will need to add assembly code to the commented handler so that it detects when an address error in load exception (exception 4) occurs and then patches register $t3 to correctly point to the array of integers. (Recall that the programmer assumed that the data segment started at address 0. Where does it actually start?)

Your modified handler should:

- Print a message indicating that the error is being handled (not ignored).
- Patch the $t3 register so that it correctly points to the "ints" array.
- Force the processor to re-execute the instruction that caused the exception (instead of skipping it, as it does now).
- NOT perform these actions on exceptions other than address error on load exceptions.

Aside: Yes, this is an odd thing for an exception handler to do: it's a very specific fix to the exact error that this buggy program exhibits. However, this is just an exercise for understanding how exceptions work.

You will know that your modified handler is correct when the average printed by the program is correct and when it only emits a *single* exception message (the first time that data from the array is loaded).

Here are a few hints and other things to remember:

- Note that the exception handling code doesn't follow the function call convention, so it doesn't save registers and doesn't have a stack. Be very careful about which registers you use, as you could clobber data from the user program. ($a0 and $v0 are safe for use. Look at how and where it saves these registers at the start of the exception handler.)
- Read the code that prints the original error message very carefully. It will show you how to extract the error number from the error register.

4 Evaluation (3 marks in total)

To earn full credit on this week's participation lab, demonstrate your updated trap handler to the TA. Be prepared to run the code, to explain how it returns to the correct line (the buggy line, rather than the line after), to state what value you chose for the start of the data segment, and how you extracted the exception's cause.

Congratulations – and best of luck finishing up your other courses this term! It's been a pleasure working with you!