CPS311: COMPUTER ORGANIZATION

Exception-Handling in a MIPS Program

The quadratic equation solver program distributed in class previously will crash with an exception if overflow occurs in computing the discriminant or the actual roots. Instructions like `add` and `sub` were deliberately used instead of `addu` and `subu` because they throw an exception on overflow The pseudo-instruction `mulo` (which the assembler expands with code to throw an exception in the case of overflow) was used instead of `mult` . This handout shows how an exception handler could be installed to catch such exceptions and handle them nicely. What is spelled out below is the additions that need to be made to the original program for this.

Add at beginning:

```
    # The following include directives contain definitions needed for
    # various symbolic names
#include <signal.h>
#include <setjmp.h>
```

Add to entry protocol, just after saving registers and before calling `compute_discriminant`

```
    /* Set up for overflow detection and recovery */
    # On MIPS, arithmetic overflow results in an exception, which is turned into a Unix
    # SIGFPE signal by the operating system.  We will install a signal handler that
    # catches this signal and use the longjmp library routine to force an abnormal exit.
    # Initialize environment with context information needed by longjmp.
    # We will use a static data area for this.
        la   $4, longjmp_env
        jal  setjmp
    # When setjmp is first called, it returns a value of 0.  If a subsequent longjmp is done
    # to unwind the stack, setjmp returns a second time with a non-zero value
        beq  $2, $0, normal_return
    # If we get here, the overflow handler has been called and has done
    # a longjmp to return - value returned by second return from setjmp
    # is status code to return to caller - already in $2
        b    fini
    # Execution resumes here after first (normal) call to setjmp
    normal_return:
    # Now install the routine handle_of as the handler for any SIGFPE (The
    # routine handle_of will be called when overflow occurs.)  The return
    # value from signal is the previous method of handling SIGFPE that was
    # in force, which must be restored before we return to the caller.
        addi $4, $0, SIGFPE      # First argument of signal is signal to be caught
        la   $5, handle_of       # address of overflow handler
        jal  signal
        sw   $2, 4($sp)          # Save old handler
```

Add to exit protocol, just before restoring $31 and destroying the stack frame

```
# Restore the old signal handler for SIGFPE.

    sw   $2, 0($sp)                # Save return value to use
    addi $4, $0, SIGFPE            # Call signal to restore old handler
    lw   $5, 4($sp)
    jal  signal
    lw   $2, 0($sp)                # Get back return value specified
```

Add the following as an additional subroutine

```
/*
 * This routine is called whenever an overflow occurs, as a result of
 * the call to signal above.
 *
 * Parameters:        $4 = name of signal
 *                    $5 = additional information about signal
 *                    $6 = address of sigcontext structure
 */

    .ent handle_of
    .frame   $sp, 0, $31

handle_of:

    # Call longjmp to do an abnormal exit

    la   $4, longjmp_env
    addi $5, $0, 3     # abnormal return code - 3 = overflow
                               # This will be in $2 upon second return
                               # from setjmp
    jal  longjmp

    # longjmp should never return, so no jr $31 needed

    .end handle_of
```

Add the following static local data section

```
# Data section - buffer needed by setjmp/longjmp

    .section .data

    .lcomm   longjmp_env, 8 * _JBLEN
```