CS/ECE 347, Spring 1996
Recitation Assignment 3: Understanding Procedure Linkages

Assigned: Tue., Jan. 30
Due: Mon., Jan. 22

1. Introduction

In this assignment, you will develop your understanding of MIPS calling and returning conventions by examining the Unix `setjmp library, which in effect cheats on the standard procedure calling conventions`

`setjmp provides a mechanism for nonlocal procedure exits` By using setjmp and longjmp, a procedure can exit to a function a long way up the procedure chain, bypassing its calling procedure.

This is typically used for error handling, in the following style:

```
jmp_buf env;

main()
{
   int jval;
   if ((jval = setjmp(env)) != 0) {
     /* Do error recovery and cleanup. */
     /* And possibly try to execute again. */
   } else {
     /* Do normal case. */
     ...
   }
}
```

The first time `setjmp() is called, it sets up the buffer env with enough information to i stack` It returns 0 to the calling function.

The calling function then does other code, and deep within some complicated hierarchy of other functions, it may find some error:

```
deep_within_some_complicated_hierarchy ()
{
  if (error_detected) {
     longjmp(env, ERROR_CONSTANT);
  }
  ...
}
```

`longjmp() restores the state that was saved in \verb`setjmp() and resets the stack pointer to the position it was in when `setjmp() was called`

/afs/cs/academic/class/15347-s96/public/recitations/ra3/longjmp.c contains a somewhat contrived example of the use of `setjmp() and \verb`longjmp(). *longjmp.mips.s* in the same directory contains an assembly version of that file as compiled by gcc 2.3.2 with the -O option.

Problem 1

The file */afs/cs/academic/class/15347-s96/public/recitations/ra3/longjmp.mips.dmp* contains a disassembly of the code for `\_setjmp() and \verb`_longjmp(), which are simplified versions of `setjmp() and \verb`longjmp(). Your assignment is to turn in an annotation of these two procedures. Show clearly what state is being saved and restored, and how longjmp subverts the MIPS calling routines to appear to 'return' to a place other than that from which it was called.

It may help you understand the code to write down what data is stored in which places in the jmp_buf. If you write this information down, turn that in too.

Things You Need To Know

This section covers information that you are not expected to know from previous lectures.

The `lwc1` operation is used to load a 32-bit value into one floating point register. An example might be

```
lwc1 \$f26, 256(\$a0)
```

This example copies the four bytes at $a0 + 256 into the floating point register $f26.

The `swc1` operation is used to store a 32-bit value from a floating point register into memory. An example might be

```
swc1 \$f26, 256(\$a0)
```

Its behavior is analogous to the behavior of normal stores.

The `cfc1` operation is used to copy the *floating point status word* to an arithmetic register. The floating point status word is a special register in the floating point coprocessor that maintains information about the behavior of the floating point unit and possible floating point exceptions. An example would be

```
cfc1 \$a0, \$f31
```

This is executed twice because the first execution forces any pending floating point exceptions to be taken. The second `cfc1` is necessary to make sure that the copy is done without any floating point exceptions.

The `ctc1` operation is used to copy an arithmetic register to the floating point status word. It is the complement of `cfc1`. It does not need to be repeated.

The value 2901261022 (0xACEDBADE) Is used as a magic token to identify a buffer that has (probably) been set by `setjmp()` A moment's thought should convince you that doing a `longjmp() to a buffer that has not been set up by \verb`setjmp() is likely to result in errors that are very difficult to track down.