MIPS videos

Over the years, Professor Zilles has made a series of videos related to MIPS programming and debugging. These will be extremely helpful for the MIPS labs and exam, so taking a look is highly recommended.

Note that “I” refers to Professor Zilles throughout this page.

Error rendering macro 'toc'


Translating C to MIPS

Here is a video showing me translate two simple C functions to MIPS, vocalizing my thought process as I write the code.

Go ahead and familiarize yourself with the C code for each function before watching the videos, since they assume familiarity with the code. The first function counts the number of times a character is found in a string; the second counts the number of times one string is found (in its entirety) in another string.

Here are complete (testable) versions of the C code and the MIPS code for both functions.

**count_letters**

Here is the C code I translate:

```c
// This function iterates through the character string "str" (which is of
// length "str_len" and counts how many instances there are of the
// character "c".
int
count_letters(char str[], int str_len, char c) {
    int count = 0;
    for (int i = 0 ; i < str_len ; ++ i) {
        if (str[i] == c) {
            count ++;
        }
    }
    return count;
}
```

Here is the resulting MIPS code:
Here is the C code I translate:

```c
int count_substring(char str[], int str_len, char sub_str[], int substr_len) {
    int count = 0;
    for (int i = 0; i < (str_len - substr_len); ++i) {
        int match = TRUE;
        for (int j = 0; j < substr_len; ++j) {
            if (str[i+j] != sub_str[j]) {
                match = FALSE;
                break;
            }
        }
        if (match) {
            count ++;
        }
    }
    return count;
}
```

Note: in my haste, I hadn't tested `count_substring` before recording this video. I had a bug in my C code. The outer loop condition `i < (str_len - substr_len)` should be `i <= (str_len - substr_len)` (note the less than or equal in the second). I discovered this in the debugging of this MIPS function as I show in the debugging videos.

Here is the resulting MIPS code:
**Caller-saved vs callee-saved registers**

In this sequence of videos I translate the following code. There are two intended goals of this demonstration:

1. demonstrating how to write a (non-trivial) function that allocates a stack frame, including managing the movement of variables to and from the stack, and
2. demonstrating the difference in usage between caller-saved and callee saved registers and when each should be used.

The demonstration has two parts: first, I do a translation using only caller-saved registers; second, I do a demonstration using callee-saved registers. Before watching the videos, familiarize yourself with the C code, since they assume familiarity with the code. Complete (testable) versions of the C code and the MIPS code will be put up soon.

Here is the C code I translate:

```c
int count_substring(char *str, char *sub_str)
{
    int str_len = strlen(str);
    int substr_len = strlen(sub_str);
    int count = 0;

    for (int i = 0; i < str_len - substr_len + 1; i++)
    {
        int match = 1;
        for (int j = 0; j < substr_len; j++)
        {
            if (str[i+j] != sub_str[j])
            {
                match = 0;
                break;
            }
        }
        if (match)
            count++;
    }

    return count;
}
```
// This function iterates through two character strings "s1" and "s2"
// performing a character-by-character comparison to see which is earlier
// in alphabetical order.
//
// Note: in order to reduce the amount of code to be translated, we've
// left out the code that checks for the end of string, so this code
// doesn't work for strings that are equal or where one string is a
// prefix of the other.
int
  case_insensitive_strcmp(const char *s1, const char *s2) {
    int i = 0;
    while (1) {
      int c1 = tolower(s1[i]);
      int c2 = tolower(s2[i]);
      if (c1 == c2) {
        ++ i;
        continue;
      } else {
        return c1 - c2;
      }
    }
  }

**Caller-saved version**

Here is the resulting MIPS code:
## here is a version using entirely caller-saved variables

```mips
# callee-saved version

Here is the resulting MIPS code:
```

```mips
## here is a version using entirely caller-saved variables

ci_strcmp1:
    sub     $sp, $sp, 20
    sw      $ra, 0($sp)
    sw      $a0, 4($sp)
    sw      $a1, 8($sp)

    li      $t0, 0          # i

ci_loop1:
    lw      $t1, 4($sp)     # s1
    add     $t1, $t1, $t0   # &s1[i]
    lb      $a0, 0($t1)     # s1[i]
    sw      $t0, 12($sp)    # save i
    jal     tolower         # c1 in $v0
    sw      $v0, 16($sp)    # save c1
    lw      $t1, 8($sp)     # s2
    lw      $t0, 12($sp)    # i
    add     $t1, $t1, $t0   # &s2[i]
    lb      $a0, 0($t1)     # s2[i]
    jal     tolower         # c2 in $v0
    lw      $t1, 16($sp)    # c1
    bne     $t1, $v0, ci_done1
    lw      $t0, 12($sp)    # i
    add     $t0, $t0, 1     # ++ i
    j       ci_loop1

    ci_done1:
    sub     $v0, $t1, $v0   # c1 - c2
    lw      $ra, 0($sp)
    add     $sp, $sp, 20
    jr      $ra
```

Callee-saved version

Here is the resulting MIPS code:
Pointers and data structures

Debugging

Here is a video showing me debug the pieces of code that I wrote in the previous videos. I had one of my TAs inject bugs into this code, so that I could demonstrate debugging, explaining the process I use. As these are relatively simple pieces of code, I don't have to pull out all of my debugging tricks; I try to start slow and pick up the pace as we move along. Although using QtSPIM is a little different from xspim, the methodology is the same.

For each video, I provide the (buggy) code version that I started with.

Example 1: count_letters

Buggy function: count_letters_1.s
Please pardon my sneeze in the above audio.

**Example 2: count_substring**

Buggy function: `count_substring_1.s`

**Example 3: count_substring**

Buggy function: `count_substring_2.s`

**Example 4: count_substring**

Buggy function: `count_substring_3.s`