CSE31 : Lab #7 – Procedures

Overview

These exercises will give you more practice with procedure calling and especially writing prologue and epilogue. Reading is in P&H 2.8.

(Exercise) Create swap1.s

Modify swap.s to translate the following procedure directly to MIPS assembly language. The `temp` variable, like all local variables in C (when not optimized), is stored on the stack. In other words you cannot use `$t0` to hold `temp`, though you may need it briefly. Hint: you will need to use 6 `lw/sw` instructions.

This exercise is slightly contrived, and could be easier if we let you optimize and use `$t0` to hold the `temp` variable, part of the point of this exercise is to see what kind of difference optimization can make.

```
void swap (int *px, int *py) {
    int temp;
    temp = *px;
    *px = *py;
    *py = temp;
}
```

(Exercise) Create swap2.s

Now modify your solution (swap1.s) to implement the following *buggy* version of the `swap` procedure.

```
void swap (int *px, int *py) {
    int *temp;
    *temp = *px;
    *px = *py;
    *py = *temp;
}
```

Q1. The bug in swap2.s is that the `temp` pointer is dereferenced without being initialized. Why might a programmer not notice this even after

testing the buggy `swap`? In other words: what situation would allow buggy swap to seem to work correctly?

(Exercise) Create swap.c

Supply the definition of a **C procedure** `proc` to be called in the main program *immediately* prior to the call to the buggy `swap` (swap2.s) that will *guarantee* that `swap` will crash when the uninitialized `temp` pointer is dereferenced (it should cause a crash on `*temp`). Also explain why your call guarantees this crash. Hint: your `proc` procedure will leave something on the stack.

```
int main () {
    int a = 1, b = 2;
    proc(/* Some args might go here */);
    swap(&a, &b);
}
```

Q2. Explain how you guarantee it crashing with what is in `proc`.

(Exercise) Fill-in nchoosek.s

This program will calculate "N choose K" entry in Pascal's triangle or the number of combinations of n distinct elements when taken k at a time. Your job is to add prologue and epilogue to complete this.

(Exercise) Create first1posv1.s

Start from first1pos.v to do the following procedure. Given a value in `$a0`, returns in `$v0` the position of the leftmost bit in the word in `$a0`. If `$a0` contains 0, store -1 in `$v0`. You are allowed to modify `$a0` in the process of finding this position. Positions range from 0 (the rightmost bit) to 31 (the sign bit).

This version should repeatedly shift left `$a0`, checking the sign bit at each shift. Work it out on paper if you are not sure how this works with a few sample numbers.

(Exercise) Create first1posv2.s

Once again start from first1pos.v to find the exact same bit as first1posv1.s. However, this version should start a mask at `0x80000000` and repeatedly shift the mask to the right to check each bit in `$a0` (without modifying the original value in $a0). You should make sure the output in version 1 and 2 are matching.

What to hand in

When you are done with this lab assignment, you are ready to submit your work. Make sure you have done the following ***before*** you press Submit:

- Answers to Q1-Q2.
- Attach swap1.s, swap2,s, swap.c, nchoosek.s, first1posv1.s and first1posv2.s
- List of collaborators