CS61c Spring 2014 Discussion 4 – MIPS Procedures

1 Overview

There are only two instructions necessary for creating and calling functions: `jal` and `jr`. If you follow register conventions when calling functions, you will be able to write much simpler and cleaner MIPS code.

2 Conventions

1. How should `$sp` be used? When do we add or subtract from `$sp`?
2. Which registers need to be saved or restored before using `jr` to return from a function?
3. Which registers need to be saved before using `jal`?
4. How do we pass arguments into functions?
5. What do we do if there are more than four arguments to a function?
6. How are values returned by functions?

When calling a function in MIPS, who needs to save the following registers to the stack? Answer "caller" for the procedure making a function call, "callee" for the function being called, or "N.A" for neither.

`$0`	`$v*`	`$a*`	`$t*`	`$s*`	`$sp`	`$ra`

Now assume our function `foo` calls another function `bar`, which is know to call some other functions. `foo` takes one argument and will modify and use `$t0` and `$s0`. `bar` takes two arguments, returns an integer, and uses `$t0`-`$t2` and `$s0`-`$s1`. In the boxes below, draw a possible ordering of the stack just before `bar` calls a function. The top left box is the address of `$sp` when `foo` is first called, and the stack goes downwards, continuing at each next column. Add "(f)" if the register is stored by `foo` and "(b)" if the register is stored by `bar`. The first one is written in for you.

1 `$ra` (f)	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

3 A Guide to Writing Functions

```
FunctionFoo: # PROLOGUE
             # begin by reserving space on the stack


             # now, store needed registers



             # BODY
             ...
             # EPILOGUE
             # restore registers



             # release stack spaces

             # return to normal execution
```

4 C to MIPS

Write an insertion sort function in MIPS that uses a swap function to accomplish the task of sorting an array of integers. The arguments to the function should be an integer array and its size. Here is the C version of the function, along with a swap helper function:

```
void swap(int * arr, int i1, int i2) {
  int t = arr[i1]; // use t <--> $t0
  arr[i1] = arr[i2];
  arr[i2] = t;
}
void insertionSort(int * arr, int size) {
  int i, j; // use i <--> $s0 and j <--> $s1
  for(i=1;i<size;i++) {
    j = i;
    while(j>0 && arr[j]<arr[j-1]) {
      swap(arr,j,j-1);
      j--;
    }
  }
}
```

A possible MIPS solution has been roughly organized on the next page.

```
        swap: # helper funcion




insertionSort: # starting point




  forLoopBody: # main for loop body

whileLoopBody: # main while loop body






 whileLoopEnd: # upon exiting while loop

   forLoopEnd: # upon exiting for loop
```