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`?

   `$sp` points to a location on the stack to load or store into. Subtract from `$sp` before storing, and add to `$sp` after restoring.

2. Which registers need to be saved or restored before using `jr` to return from a function?

   All `$s*` registers that were modified during the function must be restored to their value at the start of the function.

3. Which registers need to be saved before using `jal`?

   `$ra`, and all `$t*`, `$a*`, and `$v*` registers if their values are needed later after the function call.

4. How do we pass arguments into functions?

   `$a0`, `$a1`, `$a2`, `$a3` are the four argument registers.

5. What do we do if there are more than four arguments to a function?

   Use the stack to store additional arguments

6. How are values returned by functions?

   `$v0` and `$v1` are the return value registers.

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`
N/A	Caller	Caller	Caller	Callee	N/A	Caller

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 `$ra` (f)	5 `$t0` (f)	9 `$v0` (b)	13 `$t1` (b)
2 `$s0` (f)	6 `$ra` (b)	10 `$a0` (b)	14 `$t2` (b)
3 `$v0` (f)	7 `$s0` (b)	11 `$a1` (b)	15
4 `$a0` (f)	8 `$s1` (b)	12 `$t0` (b)	16

3 A Guide to Writing Functions

```
FunctionFoo: # PROLOGUE
             # begin by reserving space on the stack
             addiu $sp, $sp, -FrameSize

             # now, store needed registers
             sw $ra, 0($sp)
             sw $s0, 4($sp)
             ...

             # BODY
             ...
             # EPILOGUE
             # restore registers
             ...
             lw $s0 4($sp)
             lw $ra 0($sp)

             # release stack spaces
             addiu $sp, $sp, FrameSize

             # return to normal execution
             jr $ra
```

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
              sll $a1, $a1, 2    # word align i1
              sll $a2, $a2, 2    # word align i2
              addu $a1, $a0, $a1 # arr+i1
              addu $a2, $a0, $a2 # arr+i2
              lw $t0, 0($a1)     # temp  = *(arr+i1)
              lw $t1, 0($a2)     # temp2 = *(arr+i2)
              sw $t0, 0($a2)     # *(arr+i2) = temp
              sw $t1, 0($a1)     # *(arr+i1) = temp2
              jr $ra             # return

insertionSort: # starting point
              addiu $sp, $sp, -20 # push 5 words
              sw $s0, 0($sp)      # onto the stack
              sw $s1, 4($sp)      # because these
              sw $s2, 8($sp)      # will be
              sw $s3, 12($sp)     # modified
              sw $ra, 16($sp)
              move $s2, $a0       # arr
              move $s3, $a1       # size
              addiu $s0, $0, 1    # i = 1

  forLoopBody: # main for loop body
              slt $t0, $s0, $s3       # i<size
              beq $t0, $0, forLoopEnd # if false
              move $s1, $s0           # j = i

whileLoopBody: # main while loop body
              slt $t0, $0, $s1          # 0<j
              beq $t0, $0, whileLoopEnd # if false
              sll $t0, $s1, 2           # word align j
              addu $t1, $s2, $t0        # arr+j
              lw $t0, 0($t1)            # *(arr+j)
              lw $t1, -4($t1)           # *(arr+j-1)
              slt $t0, $t0, $t1         # *(arr+j)<*(arr+j-1)
              beq $t0, $0, whileLoopEnd # if false

              move $a0, $s2             # arr
              move $a1, $s1             # j
              addiu $a2, $s1, -1        # j-1
              jal swap                  # call swap
              addiu $s1, $s1, -1        # j--
              j whileLoopBody           # loop

 whileLoopEnd: # upon exiting while loop
              addiu $s0, $s0, 1 # i++
              j forLoopBody     # loop

   forLoopEnd: # upon exiting for loop
              lw $ra, 16($sp)
              lw $s3, 12($sp)
              lw $s2, 8($sp)
              lw $s1, 4($sp)
              lw $s0, 0($sp)
              addiu $sp, $sp, 20 # pop 5 words
              jr $ra             # return
```