CS61c Summer 2014 Discussion 4 – MIPS Procedures

1 MIPS Control Flow

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.

<table>
<thead>
<tr>
<th>$0</th>
<th>$v*</th>
<th>$a*</th>
<th>$t*</th>
<th>$s*</th>
<th>$sp</th>
<th>$ra</th>
</tr>
</thead>
<tbody>
<tr>
<td>(f)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(b)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Now assume a 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

1. Assuming $a0$ and $a1$ hold integer pointers, swap the values they point via the stack and return control.

   ```c
   void swap(int *a, int *b) {
       int tmp = *a;
       *a = *b;
       *b = tmp;
   }
   ```

2. Translate the following algorithm that finds the $N^{th}$ Fibonacci number to MIPS assembly. Assume $s0$ holds $N$, $s1$ holds $fib$, $t0$ holds $i$, and $t1$ hold $j$.

   ```c
   int fib = 1, i = 1, j = 1;
   if (N==0) return 0;
   else if (N==1) return 1;
   N -= 2;
   while (N != 0) {
       fib = i + j;
       j = i;
       i = fib;
       N--;
   }
   return fib;
   ```

   What must be done to make this algorithm a callable MIPS function?