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`?

   `$sp` points to a location on the stack to load or store into. Subtract from `$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 a function `foo` calls another function `bar`, which is known 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

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

```
void swap(int *a, int *b) {

  int tmp = *a;
  *a = *b;
  *b = tmp;

}
```

```
addiu $sp, $sp, -4
lw    $t0, 0($a0)
sw    $t0, 0($sp)
lw    $t0, 0($a1)
sw    $t0, 0($a0)
lw    $t0, 0($sp)
sw    $t0, 0($a1)
addiu $sp, $sp, 4
jr    $ra
```

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`.

```
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;
```

```
      beq   $s0, $0, Ret0
      addi  $t2, $0, 1
      beq   $s0, $t2, Ret1
      subi  $s0, $s0, 2
Loop: beq   $s0, $s0, RetF
      add   $s1, $t0, $t1
      addi  $t0, $t1, 0
      addi  $t1, $s1, 0
      subi  $s0, $s0, 1
      j     Loop
Ret0: addi  $v0, $0, 0
      j     Done
Ret1: addi  $v0, $0, 1
      j     Done
RetF: addi  $v0, $0, $s1
Done: jr    $ra
```

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

Add a prologue and epilogue to reserve space on the stack and store all necessary variables (see #3). Use `$a0` instead of `$s0` to store N, the function's argument.