Homework 2

Ben Lowman | CS 3330

January 7th, 2013

1. One might choose to use the add instruction as opposed to lw as the addi instruction accepts immediate values - that is, constants do not necessarily have to be stored in a register before you add them to another register. More importantly, because of its structure type ("Type I"), 16 bits are available to process the offset from a base memory address. With lw, there are only five available bits. This limits the maximum offset value to 32, which is low enough to frequently cause problems.

2. Loading the array into registers...

```
; lw is used in this case as the offset value does not exceed
    32

lw $t0 0($s3)
lw $t1 4($s3)
lw $t2 8($s3)
lw $t3 12($s3)
lw $t4 16($s3)
lw $t5 20($s3)
lw $t6 24($s3)
```

   Storing the array in reverse order...

```
sw $t6 0($s3)
sw $t5 4($s3)
sw $t4 8($s3)
sw $t3 12($s3)
sw $t2 16($s3)
sw $t1 20($s3)
sw $t0 24($s3)
```

3. Occasionally computer programs will contain more variables than the processor has registers. As such, the compiler will attempt to put less

frequently used variables into memory instead. This allows more frequently accessed variables to benefit from the faster speed of registers.

4. MIPS pseudocode is used for this problem (no corresponding registers were given)

```
sub h, h, 5
add f, g, h
```

5. Equivalent C instructions:

```
f = g + h + i;
```

6.

```
sub $t0, $s3, $s4
sll $t0, t0, 2
add $t1, $s7, $t0
lw $t2, 0($t1)
sw $t2, 32($s6)
```

7.

$$\begin{aligned} \text{0xabcdef12} &= 10 \times 16^7 + 11 \times 16^6 + 12 \times 16^5 + 13 \times 16^4 + \\ & 14 \times 16^3 + 15 \times 16^2 + 1 \times 16^1 + 2 \times 16^0 = 2882400018 \end{aligned}$$

8. Equivilant C instructions:

```
A[0] = A[1];
f = A[1] + A[0];
```

9. The instruction is an add instruction. It executes the following code:

```
add $s0, $s0, $s0
```

10. t2 has a value of 3

11.

```
fib:
       slti $t0, $a0, 2      ;continue statements
       beq $t0, $zero, cont
       addi $v0, $zero, 1    ;else make the return value 1
```

```
        jr $ra

cont:
        ;open up space in the stack
        addi $sp, $sp, -16
        sw $ra, 0($sp)
        sw $a0, 4($sp)

        ; getting n-1
        addi $a0, $a0, -1
        jal fib
        sw $v0, 8($sp)
        lw $a0, 4($sp)

        ;getting n-2
        addi $a0, $a0, -2
        jal fib
        sw $v0, 12($sp)

        ;cleaning up the stack
        lw $ra, 0($sp)
        lw $t0, 8($sp)
        lw $t1, 12($sp)
        addi $sp, $sp, 16

        ;final computation
        add $v0, $t0, $t1

        jr $ra
```