Kyler Little
CPT_S 260

Homework #3

1. Usually, there are many ways to translate a block of C code into assembly. It is always a good idea that you stick with your own way. Please translate the following code blocks into MIPS, where our convention is as follows: $S0 stores integer x; $S1 stores integer y.

(1) if (x>y+5) then x=x+y else y=x-y;

```
# PROBLEM 1 part (1)
        # Store y + 5 in $t0
        li $t0, 0
        addi $t0, $s1, 5
        # Go to label if x<=y+5
        ble $s0, $t0, ELSE
        add $s0, $s0, $s1
        j EXIT
        ELSE:
        sub $s1, $s0, $s1
        EXIT:
```

(2). while (x>y+5) {x--; x--; y++}

```
        # PROBLEM 1 part (2)
        LOOP1:
                li $t0, 0
                addi $t0, $s1, 5
                ble $s0, $t0, EXIT1     # branch to EXIT1 if x <= y+5
                addi $s0, $s0, -1
                addi $s0, $s0, -1
                addi $s1, $s1, 1
                j LOOP1
        EXIT1:
```

(3). for (x=1; x<y; x++) {x=x+5; y++}

```
        # PROBLEM 1 part (3)
        li $s0, 1
        LOOP2:
                bge $s0, $s1, EXIT2     # exit once x >= y
                addi $s0, $s0, 1
                addi $s0, $s0, 5
                addi $s1, $s1, 1
                j LOOP2
        EXIT2:
```

```
# PROBLEM 2
        # Print instruction
        li $v0, 4
        la $a0, user_instruction
        syscall

        # Read keyboard input
        li $s0, 0         # $s0 will store number of characters read so
far; if 6, then exit loop
        la $s1, arrA      # $s1 will store the address of arrA
        READ_LOOP:
                # Get input and store into array
                # 1. Where to store? Use $s2 to store address of arrA[i]
                sll $s2, $s0, 2
                add $s2, $s2, $s1    # s2 now stores address of arrA[$s0]
                # 2. Grab int from keyboard and store to $s2
                li $v0, 5      # stores result to $v0
                syscall
                sw $v0, 0($s2)  # place result in arrA[$s0] = $s2

                # Branching Condition
                addi $s0, $s0, 1
                beq $s0, 6, EXIT_READ_LOOP
                j READ_LOOP
        EXIT_READ_LOOP:
# Find second largest int in arrA
        li $t1, -32768     # $t1 will store largest int
        li $t2, -32768     # $t2 will store second largest int
        li $t3, 0          # $t3 is current index i
        LOOP_SLI:     # SecondLargestInt Loop
                # First, get address and value of arrA[i]
                sll $s2, $t3, 2
                add $s2, $s2, $s1
                lw $t4, 0($s2)  # Store value of arrA[i] into $t4

                # First check: arrA[i] > $t1
                ble $t4, $t1, CHECK1
                        # If we made it here. Then set $t2 = $t1 and $t1 =
arrA[i]
                        move $t2, $t1
                        move $t1, $t4
                CHECK1:
                # Second check: if $t1 > arrA[i] > $t2, then need to update
$t2 to arrA[i]
                ble $t4, $t2, CHECK2
                        beq $t4, $t1, CHECK2
                                move $t2, $t4
                CHECK2:

                # Branching Condition
                addi $t3, $t3, 1
                beq $t3, 6, EXIT_LOOP_SLI
                j LOOP_SLI
        EXIT_LOOP_SLI:

        # Print the second largest integer
        move $a0, $t2
        li $v0, 1
        syscall

        # Terminate the program
        li $v0, 10
        syscall
```

3. You may use either textbook or internet resources to answer the following question. Consider the instruction addi $S0, $S1, 18. Of course, I can use other constants in place of the 18. What is the minimal number I can use to replace the 18?

For the I-Type format encoding in MIPS, 16 bits of the instruction are left for the constant. Therefore, the smallest number that can replace the 18 is -32,768 or 2^{15}.