IC220: HW 2

Due: 23 Jan 2019

Full Name: _____ **Alpha:** _____

Circle Your Section: Aviv/1001 Aviv/2001 Aviv/4001 Choi/5001 Missler/5002

Preliminary: Carefully do the assigned reading for Chapter 2 (2.1-2.3,2.5-2.10,2.12)

---

1. Convert the following *pseudoinstructions* to *real* MIPS. Some notes before we start:
   - The comment, beginning with a `#`, providess the equivalent C/C++ code
   - The keyword `small` will refer to a constant value that is 16-bits wide
   - The keyword `big` will refer to a constant value that is 32-bits wide
   - The width of the constant will impact
   - Here's an example, and you can use `li` (load immediate) as part of your solutions to these questions:

     ```
     li $t1 small  # $t1 = small
     ---------------
     addi $t1, $zero, small
     ```

   - Note that `beq` and `bne` must take two registers, no constants
   - Use the temporary register `$at` as for intermediate steps as not to clobber data in other registers.
   - You can use the macros `UPPER()` and `LOWER()` to extract the upper/lower 16-bits from a constant, i.e., `big`, and use it as you would a constant, like in

     ```
     li $t1 UPPER(big)
     ```

   (a) **[2 points]** `clear $t0    # $t0 = 0`

   (b) **[2 points]** `beq $t3, small, L  # if ($t3 == small) go to L`

(c) [**4 points**] `li $t2, big  # $t2 = big; (hint, big is 32 bits!)`

(d) [**4 points**] `beq $t2, big, L  # if ($t2 == big) go to L`

(e) [**4 points**] `bge $t5, $t4, L  # if ($t5 >= $t4) go to L`

(f) [**4 points**] `lw $t0, big($t2)  # $t0 = Memory[$t2+big]`

2. For the following questions, we will map variables to registers like so:

```
f : $s0
g : $s1
h : $s2
i : $s3
j : $s4
```

For temporary registers, use `$v0` and `$v1`.

(a) [**5 points**] What is the MIPS assembly code for the following? And, did you use any pseudo-instructions? If so, draw an arrow at the instruction.

```
do{
  g = g + j;
}while( g < h );
```

(b) [**5 points**] What is the MIPS assembly code for the following? And, did you use any pseudo-instructions? If so, draw an arrow at the instruction.

```
do{
  g = g + j;
}while( g < 100 );
```

(c) [**5 points**] What is the MIPS assembly code for the following? And, did you use any pseudo-instructions? If so, draw an arrow at the instruction.

```
while( g < i){
  g = g + j;
}
```

(d) [**10 points**] What is the MIPS assembly code for the following? And, did you use any pseudo-instructions? If so, draw an arrow at the instruction.

```
while( g > i){
  g = g + 3;
}
```

3. [**15 points**] The MIPS translation of the C/C++ segment:

```
while(save[i] == k) i = i + 1;
```

is given on page 92-93 of the textbook as follows:

```
Loop: sll  $t1, $s3, 2        # t1 = i * 4
      add  $t1, $t1, $s6      # t1 = address sof save[i]
      lw   $t0, 0($t1)        # t0 = save[i] (load)
      bne  $t0, $s5, Exit     # goto Exit if save[i] != k
      addi $s3, $s4, 1        # i = i + 1
      j loop                  # goto loop body
Exit: # ... remainder of code
```

This code uses both a conditional branch and an uncodnditional jump each time through the loop. This is not efficient. **Rewrite** the assembly code so that it uses at **most one** branch or jump each time through the loop.

4. Consider the MIPS code

```
        add $t0, $zero, $zero
loop:   beq $a1, $zero, finish
        add $t0, $t0, $a0
        sub $a1, $a1, 1
        j loop
finish: addi $t0, $t0, 100
        add $v0, $t0, $zero
```

   (a) **[10 points]** Add comments to the MIPS code above. Assume that `$a0` and `$a1` are used for the input and both initially contain the integer variables `a` and `b`, which you can assume are both greater than zero. `$v0` is the return value.

   (b) **[10 points] In one sentence**, what does this code compute in terms of `a` and `b`.