HOMEWORK 6

Problem 1

- $0x00134880 = 0b000000$ 00000 10011 01001 00010 000000
  op = 0, rs = 0, rt = 19, rd = 9, shamt = 2, funct = 0

  ```
  sll $t1, $s3, 2  # Shift 0 left 2 and store it in reg t1
  ```

- $0x01364820 = 0b0000$ 0001 0011 0110 0100 1000 0010 0000
  $= 0b000000$ 01001 10110 01001 00000 100000
  op = 0, rs = 9, rt = 22, rd = 9, shamt = 0, funct = 32

  ```
  add $t1, $t1, $s6  # t1 = t1 + s6
  ```

- $0x8D280000 = 0b1000$ 1101 0010 1000 0000 0000 0000 0000
  $= 0b100011$ 01001 00000 0000 0000 0000 0000
  op = 35, rs = 9, rt = 8, imm = 0

  ```
  lw $t0, 0($t1)
  ```

- $0x15150002 = 0b0001$ 0101 0001 0101 0000 0000 0000 0010 $= 0b000101$ 01000 10101 0000 0000 0000 0010 op = 5, rs = 8, rt = 21, imm = 2

  ```
  bne $t0, $s5, 2  # Jumps to the end of the file
  ```

- $0x22730001 = 0b0010$ 0010 0111 0011 0000 0000 0000 0001 $= 0b001000$ 10011 10011 0000 0000 0000 0001 op = 8, rs = 19, rt = 19, imm = 1

  ```
  addi $s3, $s3, 1  # increments s3 (looping var)
  ```

- $0x08100004 = 0b0000$ 1000 0001 0000 0000 0000 0000 0100 $= 0b000010$ 00 0001 0000 0000 0000 0000 0100 op = 2, addr = 0x100004

  ```
  jmp 0100004  # jumps to sll
  ```

Overall, this code loops on the array until it hits a value that isn't 5. Array accessing is done on the looping var s3, which is bit-shifted by 2 bits (multiplying by 4) to get the address of each 32-bit word in the array.
At the end of the program, s3 is 4, t0 is 6, t1 is the array address + 4xs3 or 10010010, s6 is 10010000, s5 is still 5.

Problem 2

Approach here is to represent the actual resistor values in an array, and to take advantage of the fact that they are already listed in increasing order. We will therefore

- take an input val, s0
- vet the input val, if not go back
- iterate with array index var, s1 (inc by 4)
- store array val in s2
- old array item is in t2, initially it's zero
- if new array item is larger than our input value, we need to decide which is closer
- take differences of each and store them in t3 (smaller val) and t4 (larger val)
- t3 = s0 - t2
- t4 = s2 - s0
- if t3 t4, our answer is t2, otherwise it's s2

(This design was used as a reference for writing the code. Functionally, the code is the same.)

Correct code is attached. A sample output from the program can be found below (unedited) since attaching screenshots is slightly inconvenient in a latex file and this does just as well.

```
Please enter desired resistance or 0 to quit.
-134
Input cannot be negative.
Please enter desired resistance or 0 to quit.
12312321
Input cannot be larger than the largest resistor.
Please enter desired resistance or 0 to quit.
12123
You should use the resistor marked with resistance: 12000
-- program is finished running (dropped off bottom) --


Reset: reset completed.

Please enter desired resistance or 0 to quit.
14123
You should use the resistor marked with resistance: 15000
-- program is finished running (dropped off bottom) --


Reset: reset completed.

Please enter desired resistance or 0 to quit.
0
Quitting program.
```

As you can see, I said 'correct code' because it performs exactly as specified. One problem I had before it was done, is it would always print the larger value. This ended up being because I hadn't put in the logic yet for checking which one was closer (just stored the difference between lower and higher number and the desired resistance but didn't do anything with them). Once I added that it worked. This is demonstrated by the sample output - two numbers 12000 and 15000 are one after another in the array. With the input 12123, clearly closer to 12000, the program gave 12000. With the input 14123, the program gave 15000.

Future versions of this program could improve on the concept by using an arbitrary number of resistors in series to get close as possible to the input value. Even further work could guarantee an exact resistance value by using any number of resistors and resistors in parallel or in series.