1 Assembling instructions automatically

1. `slt $d, $s, $t`.

   Solution: $d$, $s$, and $t$ all fit in 32-bit signed integers since they are 5-bit unsigned ints, so we can keep them unchanged. So we can do the following:

   ```c
   int s, t, d //assume these are initialized appropriately
   //Begin with the opcode for slt, which is 101010
   int slt = 0x2A //0000 0000 0000 0000 0000 0000 0010 1010
   slt = slt | (s << 21) //0000 00ss sss0 0000 0000 0010 1010
   slt = slt | (t << 16) //0000 00ss ssst tttt 0000 0010 1010
   slt = slt | (d << 11) //0000 00ss ssst tttt dddd d000 0010 1010
   ```

   We could also make sure that register integers only contain 5-bit integers by first taking $s$ & 0x1F and similar for $t$ and $d$, but this isn’t necessary since our scanner already checks this.

2. `beq $s, $t, $i`, where $i$ is an immediate value (INT or HEXINT token).

   Solution: Proceeding similar to the above:

   ```c
   int s, t, i //assume these are initialized appropriately
   //Begin with the opcode for beq, which is 000100
   //However, we need to make sure it’s on the left 6
   //bits rather than the right 6 bits since beq is immediate-format!
   beq = 0x10000000 //0001 0000 0000 0000 0000 0000 0000 0000
   //Now add s, t in the same way as before
   beq = beq | (s << 21) //0001 00ss sss0 0000 0000 0000 0000
   beq = beq | (t << 16) //0001 00ss ssst tttt 0000 0000 0000
   //Finally, add i. We need to make sure to clear its most significant bits first!
   beq = beq | (i & 0xFFFF) //0001 00ss ssst tttt iiiii iiiii iiiii
   ```

   Note that we need to make sure to zero out the high bits of $i$ before adding it to the instruction! In what cases can we get undesirable behaviour if we forget?

How could we design our code to maximize code reuse between various instructions?

Solution: have generic functions for any register or immediate-format instruction which simply accept $s$, $t$, $d$, $f$ or $s$, $t$, $i$, $o$ respectively as arguments.