1. Translate the following C code into MIPS.

```c
// Strcpy:
// $s1 -> char s1[] = "Hello!";
// $s2 -> char *s2 = 
//    malloc(sizeof(char)*7);
int i=0;
do {
    s2[i] = s1[i];
i++;
} while(s1[i] != '\0');
s2[i] = '\0';
```

```mips
addiu $t0, $0, 0
Loop: addu $t1, $s1, $t0  # s1[i]
      addu $t2, $s2, $t0  # s2[i]
      lb $t3, 0($t1)      # char is
      sb $t3, 0($t2)      # 1 byte!
      addiu $t0, $t0, 1
      addiu $t1, $t1, 1
      lb $t4, 0($t1)
      bne $t4, $0, Loop
Done: sb $t4, 1($t2)
```

```c
// Nth_Fibonacci(n):
// $s0 -> n, $s1 -> fib
// $t0 -> i, $t1 -> j
// Assume fib, i, j are these values
int fib = 1, i = 1, j = 1;

if (n==0) return 0;
else if (n==1) return 1;
n -= 2;
while (n != 0) {
    fib = i + j;
    j = i;
    i = fib;
    n--;
}
return fib;
```

```mips
... 
Loop: beq $s0, $0, Ret0
      addu $t2, $0, 1
      beq $s0, $t2, Ret1
      addiu $s0, $s0, -2
      j Loop
Ret0: addiu $v0, $0, 0
      j Done
Ret1: addiu $v0, $0, 1
      j Done
RetF: addu $v0, $0, $s1
Done: ...
```

```c
// Collatz conjecture
// $s0 -> n
unsigned n;
L1: if (n % 2) goto L2;
goto L3;
L2: if (n == 1) goto L4;
n = 3 * n + 1;
goto L1;
L3: n = n >> 1;
goto L1;
L4: return n;
```

```mips
L1: addiu $t0, $0, 2
    div $s0, $t0      # puts (n%2) in $hi
    mfhi $t0          # sets $t0 = (n%2)
    bne $t0, $0, L2
    j L3
L2: addiu $t0, $0, 1
    beq $s0, $t0, L4
    addu $t0, $0, 3
    mul $s0, $s0, $t0
    addiu $s0, $s0, 1
    j L1
L3: srl $s0, $s0, 1
    j L1
L4: ...
```
MIPS Addressing Modes

- We have several **addressing modes** to access memory (immediate not listed):
  - **Base displacement addressing**: Adds an immediate to a register value to create a memory address (used for lw, lb, sw, sb)
  - **PC-relative addressing**: Uses the PC (actually the current PC plus four) and adds the I-value of the instruction (multiplied by 4) to create an address (used by I-format branching instructions like beq, bne)
  - **Pseudodirect addressing**: Uses the upper four bits of the PC and concatenates a 26-bit value from the instruction (with implicit 00 lowest bits) to make a 32-bit address (used by J-format instructions)
  - **Register Addressing**: Uses the value in a register as a memory address (jr)

2. You need to jump to an instruction that $2^{28} + 4$ bytes higher than the current PC. How do you do it? Assume you know the exact destination address at compile time. (Hint: you need multiple instructions)

   The jump instruction can only reach addresses that share the same upper 4 bits as the PC. A jump $2^{28} + 4$ bytes away would require changing the fourth highest bit, so a jump instruction is not sufficient. We must manually load our 32 bit address into a register and use jr.

   ```mips
   lui $at {upper 16 bits of Foo}
   ori $at $at {lower 16 bits of Foo}
   jr $at
   ```

3. You now need to branch to an instruction $2^{17} + 4$ bytes higher than the current PC, when $t0$ equals 0. Assume that we’re not jumping to a new $2^{28}$ byte block. Write MIPS to do this.

   The total range of a branch instruction is $-2^{17} + 4 \rightarrow 2^{17}$ bytes (a 16 bit signed integer that counts by words, with the PC+4 rule). Thus, we cannot use a branch instruction to reach our goal, but by the problem’s assumption, we can use a jump. Assuming we’re jumping to label Foo:

   ```mips
   beq $t0 $0 DoJump
   [...] 
   DoJump: j Foo
   ```

4. Given the following MIPS code (and instruction addresses), fill in the blank fields for the following instructions (you’ll need your green sheet!):

   ```mips
   0x002cff00: loop: addu $t0, $t0, $t0   | 0 | 8 | 8 | 8 | 0 | 0x21 |
   0x002cff04: jal foo                     | 3 |            0x00001 |
   0x002cff08: bne $t0, $zero, loop      | 5 | 8 | 0 | -3 = 0xffffd |
   ...  
   0x00300004: foo: jr $ra                 $ra=0x002cff08
   ```

5. What instruction is 0x00008A03?

   Hex -> bin: 0000 0000 0000 0000 1000 1010 0000 0011
0 opcode -> R-type: 000000 00000 00000 10001 01000 000011
sra $s1 $0 8