Name:_____

**Problem 1: (15 points) Floating Point Representation=**

1. **(5 points)** Translate the decimal number -5.375 to a 32 bit IEEE 754 single precision floating-point number

2. **(5 points)** Translate the hexadecimal number `0x3CC00000` to decimal number in normalized scientific notation form (3 points partial credit if in normalized binary scientific notation).

3. **(5 points)** Translate the hexadecimal number `0x021FC027` to a MIPS instruction using register names, e.g. $s0, $t0, …

Name:_____

**Problem 2: (30 points) MIPS Reverse Compilation**

Consider the following MIPS assembly code:

```
          add  $v0, $zero, $zero
          add  $v1, $zero, $zero
          add  $s0, $zero, $zero
Loop:     add  $s1, $s0, $s0
          add  $s1, $s1, $s1
          add  $t0, $a0, $s1
          add  $t1, $a1, $s1
          lw   $t4, 0($t0)
          slt  $s4, $zero, $t4
          bne  $s4, $zero, There
          sub  $t4, $zero, $t4
          addi $v1, $v1, 1
          j    Next
There:    addi $v0, $v0, 1
Next:     sw   $t4, 0($t1)
          addi $s0, $s0, 1
          bne  $s0, $a2, Loop
Exit:     ...
```

1. **(10 points)** Assume that two arrays A and B are located somewhere in memory; the base addresses of A and B are stored in `$a0` and `$a1`, respectively, and their size is stored in `$a2`. Describe concisely what the code does. Specifically, at the end of execution, what will be stored in array B? What values will `$v0` and `$v1` contain?

Name:_____

2. **(20 points)** Convert the following instructions from the code above into 32 bit hexadecimal number. Assume that the address of the first instruction (`add $v0, $zero, $zero`) is located at address `0x04000000`.
   a. **(5 points)** `add $t0, $a0, $s1`

   b. **(5 points)** `lw $t4, 0($t0)`

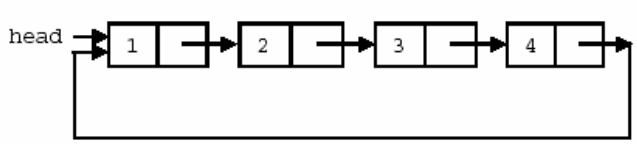
   c. **(5 points)** `j Next`

   d. **(5 points)** `bne $s0, $a2, Loop`

Name:_____

**Problem 3: (40 points) Circular Lists**

We're writing a circular linked list to keep numbers. The idea is very similar to the single-linked list we discussed in class, but the last element will now point to the first element, instead of having a NULL pointer. Our circular linked list is made up of elements of type `pair`. Here's an example of a circular linked list on the left, with the `pair` structure definition on the right:




The `pair` structure is defined as follows:

```
struct pair {
    int car;          // "number"
    struct pair *cdr; // next "pair"
} *head;
```

In the above figure, we have 4 pair structures linked into a circular list. The values of the circular list are { 1, 2, 3, 4 }. The variable head is a pointer to a pair structure and it always points to the first element of the list.

1. **(10 points)** Given the following data layout in memory and knowing that the variable `head = 0x1000`, draw a sketch similar to the one above of the circular list represented below:

| Address | Value |
|---|---|
| 0xFFFF | . . . |
| | 0x8480 |
| 0xFFA4 | 0x000F |
| | . . . |
| | 0xFFA4 |
| 0xC84C | 0x0008 |
| | . . . |
| | 0x1000 |
| 0x8480 | 0xC84C |
| | . . . |
| | 0x0000 |
| 0x1000 | 0x2480 |
| | . . . |
| | 0xC84C |
| 0x0000 | 0x0004 |

Name:_____

2. **(25 points)** Below is a function called `reset_numbers` that attempts to set all the values in the circular list to the specified integer.

```
void reset_numbers(struct pair *p, int i)
{
        if( p != NULL) {
             p->car = i;
             reset_numbers(p->cdr, i);
        }
}
```

Convert `reset_numbers` to MIPS assembly. You must exactly translate the code above, i.e. you should not try to optimize it and it must be recursive. Also, you must follow all of the MIPS procedure conventions. Failure to do either of these will result in a significant loss of points.

3. **(5 points)** In one sentence, describe what happens on an actual MIPS machine if we call `reset_numbers` on a non-empty list as described in this problem