MIPS Control Flow

1) What are the instructions to branch on each of the following conditions?

<table>
<thead>
<tr>
<th>Condition</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s0 &lt; $s1</td>
<td></td>
</tr>
<tr>
<td>$s0 &lt;= $s1</td>
<td></td>
</tr>
<tr>
<td>$s0 &gt; 1</td>
<td></td>
</tr>
<tr>
<td>$s0 &gt;= 1</td>
<td></td>
</tr>
</tbody>
</table>

2) Complete the MIPS so that it flows like the 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’;

addi $t0, $0, 0
Loop: add $t1, $s1, $t0  # s1[i]
add $t2, $s2, $t0  # s2[i]
lb $t3, 0($t1)  # char is
sb $t3, 0($t2)  # 1 byte!
addi $t0, $t0, 1
addi $t1, $t1, 1
________________
________________
Done: add $t2, $t2, 1
sb $t4, 0($t2)

// Nth Fibonacci(N):
// $s0 -> N, $s1 -> fib
// $t0 -> i, $t1 -> j
// assume the following values
// are in registers already
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;

subi $s0, $s0, 2
Loop: add $s1, $t0, $t1
add $t0, $t1, 0
add $t1, $s1, 0
subi $s0, $s0, 1
j Loop
Ret0: add $v0, $0, 0
j Done
Ret1: add $v0, $0, 1
j Done
RetF: add $v0, $0, $s1
Done: jr $ra
MIPS Functions
Translate the given C code into MIPS and vice versa.

```c
void swap(int *a, int *b) {
    int temp = *a;  // use the stack
    *a = *b;
    *b = temp;
}

/* What does this program do? */
int Mystery(unsigned int a) {
    Mystery:  addi  $a1, $0, 0
               addiu $sp, $sp, -4
               sw    $ra, 0($sp)
               jal   Recur
               lw    $ra, 0($sp)
               addiu $sp, $sp, 4
               jr    $ra

    Recur:   bne   $a0, $0, Body
             add $v0, $0, $0
             jr    $ra

    Body:    addi  $a1, $a1, 1
              srl   $a0, $a0, 1
              addiu $sp, $sp, -4
              sw    $ra, 0($sp)
              jal   Recur
              addi $v0, $v0, 1
              lw    $ra, 0($sp)
              addiu $sp, $sp, 4
              jr    $ra

int strcmp(char *s1, char *s2) {
    do {
        int cc = charcmp(*s1, *s2);
        if(cc) {
            return cc;
        }
        s2++;
    } while(*s1++);
    return 0;
}
```

```
Foo:  add  $v0, $zero, $zero
Loop: slti $t0, $a1, 1
      bne  $t0, $zero, End
      sll  $t1, $a1, 2
      add  $t2, $a0, $t1
      lw   $t3, 0($t2)
      add  $v0, $v0, $t3
      addi $a1, $a1, -1
      j    Loop
End:  jr   $ra
```
What are the 3 meanings unsigned can have in MIPS?

MIPS Pseudoinstructions

MIPS and MARS have a number of convenient instruction names that are not part of the actual MIPS instruction set. These are converted into MIPS instructions for you by the assembler, such as move $v0, $a0 --> add $v0, $a0, $0. Convert the following:

<table>
<thead>
<tr>
<th>R-Format</th>
<th>I-Format</th>
<th>J-Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>not $v0, $t0</td>
<td>b gt $s0, $0, Label</td>
<td>li $s0 0x8800FF</td>
</tr>
</tbody>
</table>

MIPS Instruction Formats

Instructions are represented as bits, same as everything else! All instructions fit in a word (32 bits). In order to cover all the different instructions, there are 3 different instruction types:

- **R-Format**
  - opcode(6) | rs(5) | rt(5) | rd(5) | shamt(5) | funct(6)

- **I-Format**
  - opcode(6) | rs(5) | rt(5) | immediate(16)

- **J-Format**
  - opcode(6) | target address(26)

Here the instruction formats are written out by field name and width in bits in parentheses. The first bit of the opcode is the MSB and the other end is the LSB. The fields are named as follows:

- **opcode** - Picks family of operation
- **rs** - First register source
- **rt** - Second register source
- **rd** - Destination register
- **shamt** - Shift amount
- **funct** - Variant of operation
- **immediate** - Relative address or constant
- **address** - Absolute address

The exact instruction is determined by the opcode along with the funct, if applicable. While the format names don’t actually stand for anything, it’s easy to remember them as R for register or regular, I for immediate, and J for jump.

Based on the format type names and the field names and descriptions, **what format should the following instructions take?** Answer R, I, or J.

- sub ____
- addi ____
- xori ____
- srl ____
- sltiu ____
- lw ____
- mult ____
- jal ____
- jal ____

**add** and **sub** share the same opcode. Why is there no **subi** instruction?