MIPS cheat sheet

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>add dest, src0, src1</td>
<td>add $s0, $s1, $s2</td>
</tr>
<tr>
<td>sub</td>
<td>sub dest, src0, src1</td>
<td>sub $s0, $s1, $s2</td>
</tr>
<tr>
<td>addi</td>
<td>addi dest, src0, immediate</td>
<td>addi $s0, $s1, 12</td>
</tr>
<tr>
<td>lw</td>
<td>lw dest, offset(base addr)</td>
<td>lw $t0, 4($s0)</td>
</tr>
<tr>
<td>sw</td>
<td>sw src, offset(base addr)</td>
<td>sw $t0, 4($s0)</td>
</tr>
<tr>
<td>beq</td>
<td>beq src0, src1, branchAddr</td>
<td>beq $t0, $t1, Eq</td>
</tr>
<tr>
<td>bne</td>
<td>bne src0, src1, branchAddr</td>
<td>bne $t0, $t1, notEq</td>
</tr>
<tr>
<td>j</td>
<td>j jumpAddr</td>
<td>j jumpWhenDone</td>
</tr>
</tbody>
</table>

// $s0 -> a, $s1 -> b
// $s2 -> c, $s3 -> z

int a=4, b=5, c=6, z;
z = a+b+c+10;

// $s0 -> int *p = (int *)malloc
//                  (3*sizeof(int));

// $s1 -> a
p[0] = 0;
int a = 2;
p[1] = a;
p[a] = a;

// $s0 -> a, $s1 -> b
int a = 5, b = 10;
if (a + a == b) {
    a = 0;
} else {
    b = a - 1;
}

/*What does this do? (Not C, in English) */

Implement streq, which sets $v0 to true if its two character pointer arguments ($a0 and $a1) point to equal strings (and false otherwise), in MIPS (there is not enough room to do this on this sheet of paper; use another).
What are the instructions to branch on each of the following conditions?

$s0 < s1$

$s0 <= s1$

$s0 > 1$

$s0 >= 1$

What are the 3 meanings unsigned can have in MIPS?

**Pointer Exercises**

1) Given the following definition of a binary tree, implement a binary search routine for the tree.

```c
struct btree {
    int value;
    struct btree
        *left, *right;
};

// return true iff x is in the tree.
// Assume left branch is "less".
int bsearch(struct btree *t, int x) {
}
```

2) What will C print? Assume the following. The `%p` format specifier prints a pointer.

```c
int arr[9]; // lives at address 0x10000000

printf("%p\n",arr); printf("%p\n",arr+1);
printf("%p\n",&arr[0]); printf("%p\n",&arr[0]+1);
```