Homework 5

1. Consider these two "instructions", which are not really part of the MIPS language:

`andn $t1, $t2, $t3`	calculates the bitwise AND of `$t2` and `$t3`'s bitwise complement
`xnor $t1, $t2, $t3`	calculates the bitwise complement of `$t2` XOR `$t3`

A. For each instruction above, if `$t2` holds 0x00FFA5A5 and `$t3` holds 0xFFFF003C, what will `$t1` hold after execution? *(0x0000A581; 0x00FF5A66)*

B. If these were implemented as pseudo-instructions, they would be replaced with one or more real instructions upon assembly. What instructions might they be replaced with?

C. Why are these instructions probably not part of the MIPS language? If they were, what type would you expect them to be, and why? *(R-type; R-type)*

2. Consider the two lines of C code below. (The first uses the bitwise-AND operator, and the second uses the question mark operator. Search for "c question mark operator" if you aren't familiar with the latter in C and Java.) You may assume that all the values are `int`s.

```
value1 = value2 & array[0];
value1 = value1 ? value2 : array[0];
```

A. For these statements, assume that `value1` holds 0x00000000, `value2` holds 0x00002222, and `array[0]` holds 0x00001234. What value will `value1` hold after each line has executed (independently of the other)? *(0x00000220; 0x00001234)*

B. For each statement, write a minimal sequence of MIPS instructions that performs the same task. You may assume that the values `value1`, `value2`, and `array` (but not `array[0]`) are already stored in registers.

3. For each part, answer the question assuming that the register `$t0` holds the values 0xAD100002 and 0xFFFFFFFF in turn.

A. If `$t1` contains 0x3FF80000 and `$t0` holds the values as stated above, what is held in `$t2` after the code at right executes each time? *(0x00000001; 0x00000001)*

```
        slt $t2, $t0, $t1
        beq $t2, $0, else
        j done
else:   addi $t2, $0, 2
done:   ...
```

B. Let the value in `$t0` be compared against some immediate value X in the instruction `slti $t2 $t0, X`. For what values of X (if any) will `$t2` be set? *(all values; 0x0000-0x7FFF)*

C. Suppose the program counter (`PC`) is currently set to 0x00000020. Is it possible to jump to the values in `$t0` using one or more `j` instructions? Is it possible with one or more `beq` instructions? *(neither are possible; neither are possible)*

4. For each part, let `$t0` contain the values 0x00001000 and 0x20001400 in turn.

A. If `$t0` holds the values as stated above, what is held in `$t2` after the code at right executes each time? *(0x00000000; 0x00000000)*

```
        slt $t2, $t0, $t0
        bne $t2, $0, else
        j done
else:   addi $t2, $t2, 2
done:   ...
```

B. If `$t0` holds the values as stated above, what is held in `$t2` after the code at right executes each time? *(0x00000000; 0x00000001)*

```
sll $t0, $t0, 2
slt $t2, $t0, $0
```

C. Suppose the program counter (`PC`) is currently set to 0x20000000. Is it possible to jump to the values in `$t0` using one or more `j` instructions? Is it possible with one or more `beq` instructions? *(`beq` will work, but `j` will not (practically); both will work)*

5. Consider the addresses 0x00001000 and 0xFFFC0000.

A. If the `PC` is at address 0x00000000, how many branch instructions are needed to get to each of the addresses above without using any jump instructions? *(1 branch; 3 branches)*

B. If the `PC` is again at address 0x00000000, how many jump instructions are needed to get to each of the addresses above without using any branch instructions? *(1 jump; not practically possible)*

C. Redo part A, but assuming that we are using a MIPS variant that has 8-bit immediates in I-type instructions, rather than 16-bit immediates. *(8 branches; 517 branches)*

6. For these three questions, we are altering the format of MIPS instructions in some way. State what the effect on the given instruction would be. You should assume that every instruction remains 32 bits long, with the other fields (except the opcode) growing or shrinking as necessary. Explain all your answers.

A. How would the `beq` instruction differ if there were only 8 registers, rather than 32? How would it differ if the immediate field were only 10 bits long? *(16× larger range; 64× smaller range)*

B. How would the `j` instruction differ if there were only 8 registers? If the address field were only 10 bits long? *(no effect; `PC` & target address must share high 20 bits)*

C. How would the `jr` instruction differ if there were only 8 registers? If the address field were only 10 bits long? *(no effect except restricted choice; no effect)*