Problem 1: The MIPS I bgtz and bltz instructions compare a register to zero, but can’t compare two registers (unless the second one is the zero register). Consider an extension of MIPS I that allowed branch greater than and branch less than instructions to compare two registers, call the new instructions bgt and blt. Explain why the existing bgtz opcode could be used for bgt but why the bltz opcode could not be used for blt. Hint: See bltzal.

Problem 2: A C function and a part of a MIPS equivalent are shown below. The C function looks at the attributes of a car and decides what to pack in a promotional giveaway to the car buyer. The assembler code corresponds to the C function up until the last line (checking for a sun roof).

```c
#define FE_SPORTY 0x1
#define FE_OFF_ROAD 0x2
#define FE_EFFICIENT 0x4
#define FE_SUN_ROOF 0x10000
#define FE_MANUAL_TRANSMISSION 0x20000
enum Giveaways { G_Food, G_Hiking_Boots, G_Sunblock, G_Driving_Gloves }; void prepare_promotion_package(Car_Object *car) { int car_features = car->features;
  if ( car_features & FE_OFF_ROAD ) pack(car, 1200, G_Hiking_Boots);
  if ( car_features & FE_SUN_ROOF ) pack(car, 200, G_Sunblock);
}
```

```mips
# MIPS-I Equivalent of C code.
#
#    $a0: Address of car object.
#    Notes: Procedure call arguments placed in $a0, $a1, ...
#    Assume that pack does not change $a0-$a3 or $s0-$s7

lw $s0, 16($a0)       # Load the features bit vector of car object.
andi $t0, $s0, 2
beq $t0, $0, SKIP1
addi $a1, $0, 1200
jal pack
addi $a2, $0, 1

SKIP1:
#
# PART b SOLUTION STARTS HERE

(a) The MIPS code above omits the last line of C code (checking for a sun roof); complete it using MIPS I instructions. (Do this on paper, there is no need to run it.) Hint: A clever solution uses five instructions a straightforward solution uses six instructions. If you have more than ten instructions ask for help.
```
(b) Add comments to the assembler code above. Write the comments for an experienced MIPS and C programmer, that is, the comments should describe what an instruction is doing in terms of what the C code is trying to do. The comments should not just describe how instructions change register values.

For example, a bad comment for the lw instruction would be: Compute address 16 + $a0, retrieve word starting at that address and write into $s0. This is a bad comment because an experienced MIPS programmer already knows what an lw instruction does. The comment for lw in the code (Load the features...) is good because it tells the reader what the $s0 value is in terms of what the code is supposed to do.

**Problem 3:** Consider the code from the previous problem. Invent a new branch instruction that can be used for the kind of branching used in the code: testing if a single bit in a register value is 1.

(a) Show the encoding for the new branch instruction. The new instruction must fit as naturally as possible with other instructions.

(b) Compare the implementation cost and performance of the new instruction to the existing MIPS-I bltz and to a hypothetical blt instruction. (With each instruction doing its own thing, not as part of functionally equivalent alternatives.)

**Problem 4:** Solve Fall 2007 Homework 2 without looking at the solution. Then look at the solution and give yourself a grade on a scale of [0, 1]. **Warning:** test questions are based on the assumption that homework problems were completed, so make a full effort to solve it without first consulting the solution.