LSU EE 4720 **Homework 1** Solution **Due: 15 September 2010**

Problem 1: Diagnose or fix the MIPS-I problems below.

(*a*) Explain why the code fragment below will not complete execution. Fix the problem, assuming that the load addresses are correct. (Problems such as this occur when operating on data prepared on a different system.)

```
lw r1, 0(r2)
lw r3, 6(r2)
```

MIPS loads and stores must be to aligned addresses, meaning that the address must be a multiple of the data size. In this case the data size is 4 (because the instructions are `lw`). Because they both use the same base register, `r2`, at most one of the load addresses can be a multiple of 4.

The code below fixes the problem by using `lb` instead of `lw` and sliding the bytes into the respective destination registers. A faster solution is possible: based on the two least-significant bits of `r2` and branch to one of four routines. (For example, if the two-least significant bits were zero then `lw r1, 0(r2)` would work and two `lhu` could be used for `6(r2)`.)

```
# Solution

 addi r4, r2, 4
LOOP:
 sll r1, r1, 8
 lbu r11, 0(r2)
 or r1, r1, r11
 sll r3, r3, 8
 lbu r13, 6(r2)
 or r3, r3, r13
 bne r2, r4 LOOP
 addi r2, r2, 1
```

(*b*) The code below will execute, but it looks like there might be a bug. Explain.

```
jal subroutine
add r31, r0, r0
```

The `jal` instruction writes the return address in register `r31`, but the instruction in the delay slot, which is executed immediately after the `jal`, overwrites `r31`. If the programmer didn't care about the return address then a `j` instruction would be used, so the code above probably has a bug.

(*c*) The two fragments below are almost but not quite MIPS-I. Re-write them using MIPS instructions so they accomplish what the programmer likely intended.

```
# Fragment 1
lw r1, 0(r2+r3)

# Fragment 2
bgti r1, 101  target
nop
```

```
# Solution - Fragment 1
#
# MIPS does not have a load that uses two source registers.
add r1, r2, r3
lw r1, 0(r1)

# Solution - Fragment 2
#
# MIPS branches cannot perform magnitude comparisons (gt between two
# registers) nor can they compare to an immediate (the 101).
slti r2, r1, 102
bne r1, r0 target
nop
```

(*d*) The code fragments below are correct, but not as efficient as they could be. Re-write them using fewer instructions (and without changing what they do).

```
# Fragment 1
addi r1, r0, 0xaabb
sll  r1, r1, 16
ori  r1, r1, 0xccdd

# Fragment 2
add r1, r0, r0
addi r1, r1, 123
```

```
# Fragment 1 - Solution
#
# Hel-llow, lui is in the instruction set for a reason!
lui r1, 0xaabb
ori r1, r1, 0xccdd

# Fragment 2 - Solution
#
# Too much exposure to accumulator-style ISAs can ingrain bad habits.
# In particular a register does not need to be cleared before it is
# written.
addi r1, r0, 123
```