LSU EE 4720 **Homework 3** **Due: 11 March 2015**

Problem 1: *For the following question refer to the Intel 64 and IA- 32 Architectures Software Developer's Manual linked to the course references page.* Intel 64 is an example of a CISC ISA, but not a good example because it evolved from an ISA designed for a 16-bit address space. Over the years the size of the general purpose registers increased from 16 bits to 64 bits and the number of general-purpose registers increased from 8 to 16.

(*a*) Show the 64-bit names of the general purpose registers provided by Intel 64. (See Chapter 3 of the manual mentioned above.)

(*b*) A MIPS assembly language instruction uses the same name for a register regardless of how many bits of the register we use. For example, `sb r1, 0(r2)` uses 8 bits of `r1` and `sw r1, 0(r2)` uses all 32 bits, but in both instructions we refer to `r1`. Not so in IA-32/Intel 64, in which the name of the register indicates how many bits to use. Show the names for `RAX` for the different sizes and positions in the register.

Problem 2: *Diagrams of the MIPS implementation for this problem can be found in EPS format at* `http://www.ece.lsu.edu/ee4720/2015/hw02-p3-if-mux-sol.eps` *and in Inkscape SVG (which can easily be edited) at* `http://www.ece.lsu.edu/ee4720/2015/hw02-p3-if-mux-sol.svg`.

As has been pointed out in class, MIPS lacks a `bgt rs, rt, TARG` (branch greater than comparing *two* registers) instruction because the ISA was designed for a five-stage implementation in which the branch is resolved in `ID`. To resolve `bgt` in `ID` one would have to compare two register values starting about half-way through the cycle, something that might slow down the clock frequency.

In this problem suppose there was a `bgt` instruction in MIPS. We would like the implementation to have the same clock frequency as our `bgt`-less implementation. One way of doing that is by resolving `bgt` in `EX` (but still resolving the other branches in `ID` as they are now). If we resolve in `EX` we can expect a one-cycle branch penalty, as can be seen in the PED below.

```
# Cycle             0  1  2  3  4  5  6  7
 bgt r1, r2, TARG   IF ID EX ME WB
 add r3, r4, r5        IF ID EX ME WB
 xor r6, r7, r8           IF IDx
 ...
TARG:
 lw r9, 0(r10)                IF ID EX ME WB
# Cycle             0  1  2  3  4  5  6  7
```

Note that `xor` is squashed in cycle 3, which is the behavior we want for a taken `bgt` (see the second subproblem below). If `bgt` were not taken then no instruction would be squashed.

(*a*) Modify the implementation on the next page (taken from the Homework 2 solution) so that `bgt` is resolved in `EX`. *Note: The original assignment had a very big typo in the previous sentence: giving* `ID` *instead of* `EX` *as the stage to resolve in.*

- Pay attention to cost. Assume that a magnitude comparison (*e.g.*, greater than) is relatively costly.
- Show the control logic for `bgt`.
- Do not "break" existing instructions.

(*b*) If `bgt` is taken an instruction will have to be squashed. (Because `bgt` has just one delay slot, just like all the other branches.) Add logic so that a one-bit signal `sq` (squash) is delivered to `ID` when the instruction in `ID` needs to be squashed due to a taken `bgt`.