LSU EE 4720 **Homework 2** Solution **Due: 27 February 2015**

For those preparing electronic submission of a solution (E-mail) and who would like a vector-format version of the MIPS implementation can find it in Encapsulated Postscript at `http://www.ece.lsu.edu/ee4720/2015/mpipei3.eps` *and for those who would like to edit the image can find it in Inkscape SVG at* `http://www.ece.lsu.edu/ee4720/2015/mpipei3.svg`.

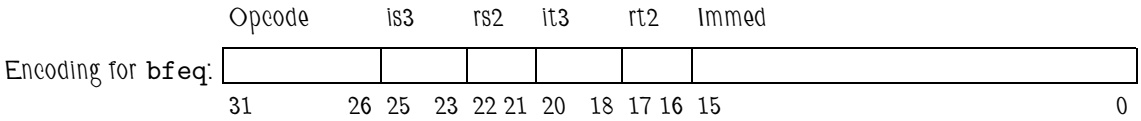
Problem 1: *The following problem appeared on the Spring 2014 Final Exam as Problem 7c.* Suppose the 16-bit offset in MIPS `lw` instructions was not large enough. Consider two alternatives. In alternative 1 the offset in the existing `lw` instruction is the immediate value times 4. So, for example, to encode instruction `lw r1, 32(r2)` the immediate would be 8. In alternative 2 the behavior of the existing `lw` is not changed but there is a new load `lws r1, 32(r2)`, in which the immediate is multiplied by 4. Note that alternative 2 requires a new opcode. Which instruction should be added to a future version of MIPS, alternative 1 or alternative 2? Explain.

Alternative 2, `lws`, should be chosen so that existing software continues to run correctly.

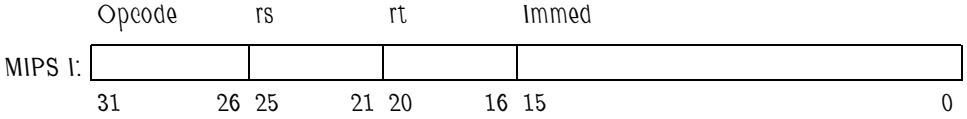
Problem 2: *The following problem appeared on the Spring 2014 Final Exam as Problem 5.* The displacement in MIPS branches is 16 bits. Consider a new MIPS branch instruction, `bfeq rsn, rtn` (branch far), where `rsn` and `rtn` are 2-bit fields that refer to registers 4-7. As with `beq`, branch `bfeq` is taken if the contents of registers `rsn` and `rtn` are equal. With six extra bits `bfeq` can branch 64 times as far.

(*a*) Show an encoding for this instruction which requires as few changes to existing hardware as possible. Explain how your choice of encoding minimizes changes.

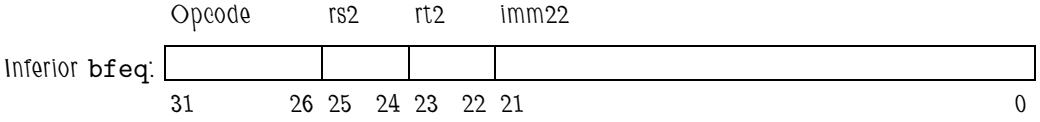
The encoding for `bfeq` is shown below. The 22-bit displacement is `is3,it3,Immed` (in Verilog notation).



In the encoding above the `rs` and `rt` register fields each have been shortened from five to two bits. (The original format I encoding is shown below.) Since the fields have been shortened but not moved the four remaining bits can be connected directly to the register file. (See the solution to the next part.)



With the encoding above we need two 3-bit multiplexors at the register file address inputs (see the next subproblem). The encoding below, which would receive partial credit, would require two 5-bit multiplexors at the register file address inputs and so would be more costly. This inferior encoding is certainly more organized with `rs2` and `rt2` next to each other and with the immediate occupying 22 contiguous bits. However the multiplexors at the register file inputs would need to be five bits instead of three bits. Notice that it does not make a difference whether or not the immediate bits are contiguous, as they are below, or split, as they are in the solution above.

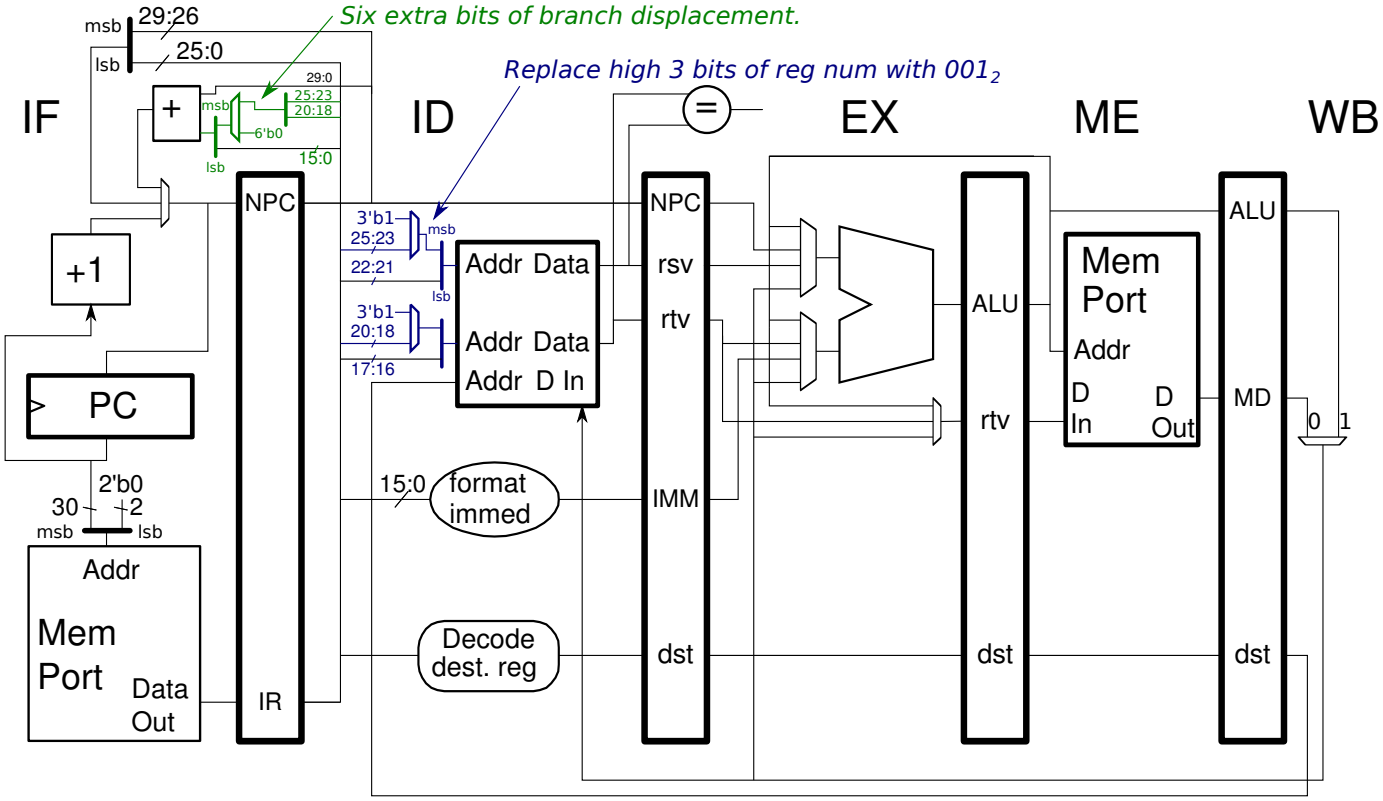


Grading Note: Despite the hint about cost, almost no one used the preferred solution, the one with the immediate field split.

(*b*) Modify the pipeline below to implement the new instruction. Use as little hardware as possible.

Changes appear below. At the register file address inputs, shown in blue, the high three bits of each register number is determined by a multiplexor. If a `bfeq` is present the upper inputs are used, making the upper three bits of each register number 001_2, otherwise the upper bits come from the instruction. The lower two bits are taken from the instruction regardless of whether `bfeq` is present. If a `bfeq` is present the displacement includes the extra six bits, these changes are shown in green.

Grading Note: Many solutions did show logic selecting a larger displacement size, but that larger displacement was sent to the `EX` *stage as an immediate rather than to the adder feeding the* `IF`*-stage multiplexor. We are proud of our resolve-branch-in-*`ID` *pipeline, so of course points must be deducted when this feature is ignored. Better to have points deducted on a homework than on a test.*



Problem 3 on next page.

Problem 3: Show the control logic for the IF-stage multiplexor in the MIPS implementation below.

- The control logic should work for `beq`, `bne`, `bgtz`, `bgez`, and `j`. Assume that any other instruction is not a control transfer.
- Show exactly which `IR` bits are needed by the control logic that detects `bgez` *(Hint, hint.)* and other instructions.
- The control logic should check the condition to determine if the branch is taken.
- Pay attention to cost.

Solution appears below in blue. The input numbers on the `IF`-stage mux were chosen so that the input for a taken branch (shown as `t-br`) is 01_2, the input for a jump is 10_2, and the input for the incremented PC (`inc`) is 00_2.

The logic detecting a `j` is connected directly to the most-significant bit of the multiplexor select input. The logic detecting each of the four kinds of branches (which obviously is not a complete list) connects to an AND gate which detects whether the respective condition is true.

The full opcode for the `bgez` instruction is in the `opcode` and `rt` fields. (The `rt` field for the `bgez` plays the same role as the `func` field for the type R instructions.) For the jump and the other branches listed it is sufficient to only look at the opcode field.

A lower-cost solution would use just one comparison unit with a mux at the lower input selecting either `rtv` or the constant 0. That would make the critical path in `ID` a bit longer.