LSU EE 4720 Homework 3 Due: 20 February 2019

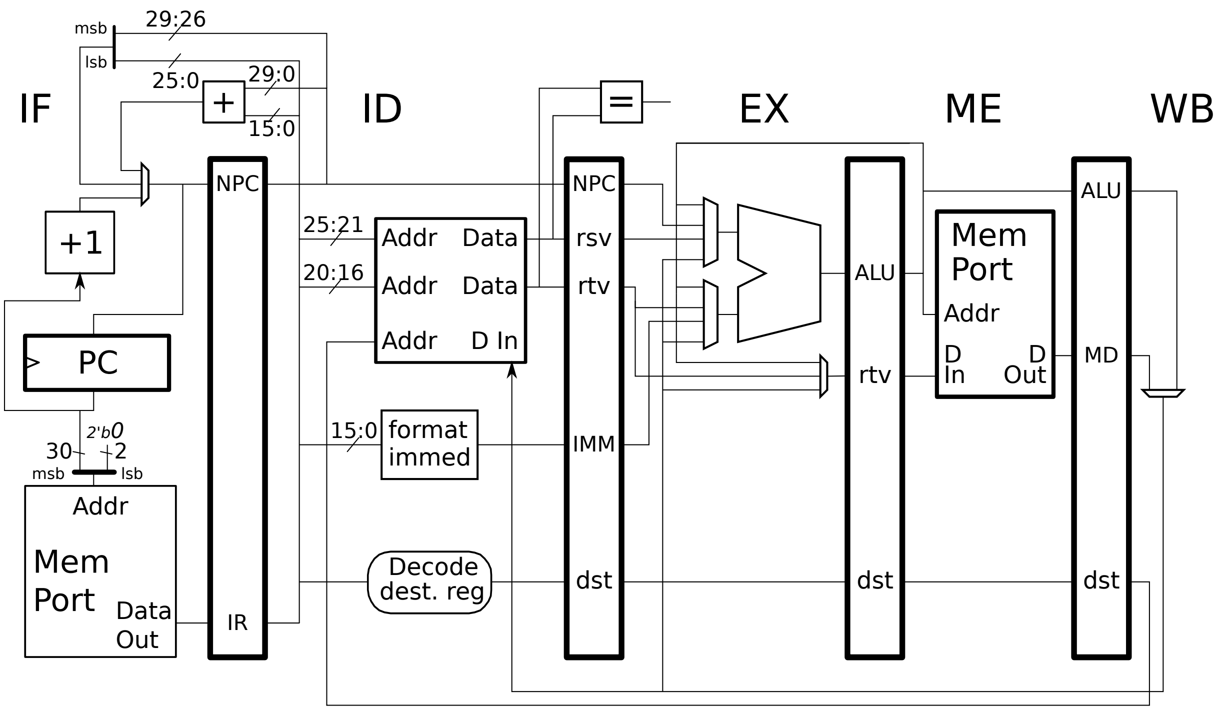
Before solving the branch hardware problem below it might be helpful to look at 2016 Homework 2.

Problem 1: The code below should suffer a stall on the illustrated implementation due to a dependency between the `addi` and `bne` instructions. The stall can be avoided by scheduling the loop, but lets consider a hardware solution for code fragments like this in which an `addi rX, rY, IMM` is followed by a `bne rX, r0, T` or by a `beq rX, r0, T`.

```
LOOP:
 addi r3, r3, -1
 bne r3, r0, LOOP
 lw r1, 4(r1)
```

One way to avoid the stall (which would work for more than just the cases outlined above) would be to have the ALU generate an `=0` signal which, if the dependencies were right, could be used by the branch hardware. Alas, the ALU people are on vacation, so lets try something else.

As alert students may have realized by now, all the branch hardware has to do is check whether `rY == -IMM`, which is `r3 == 1` in the example. The comparison itself can be done using the existing comparison logic. The challenge is delivering the operands to that logic at the right time.



Attention students who have forgotten how to use a pencil (or never learned): An Inkscape SVG version of the implementation can be found at `https://www.ece.lsu.edu/ee4720/2019/mpipei3.svg`.

(*a*) Add hardware to the implementation above to deliver the correct operands to the comparison unit so code fragments like the one above can execute without a stall.

- Pay attention to cost, including the number of bits in each wire used. (For example, don't add a second comparison unit.)
- The changes should not prevent other code from executing correctly. (For example, a branch such as `beq r1,r2, T` should execute correctly.)

- Don't overlook that `rX` and `rY` are not necessarily the same register.

(*b*) Add control logic to generate a `BY` signal which is set to logic `1` when the branch can use the bypass. The control logic must detect that the correct instructions (including the registers) are present.

(*c*) If the design above was done correctly the highest cost part is the logic handling the immediate. Show how the cost of that logic can be reduced while still retaining most (but not all) of the benefits of the full-cost design. Your argument should include examples of "typical" code. (Assume [actually assert] that your code samples are typical [reflects what is running by users most of the time]. Later in the semester we'll remove the scare-quotes from "typical".)