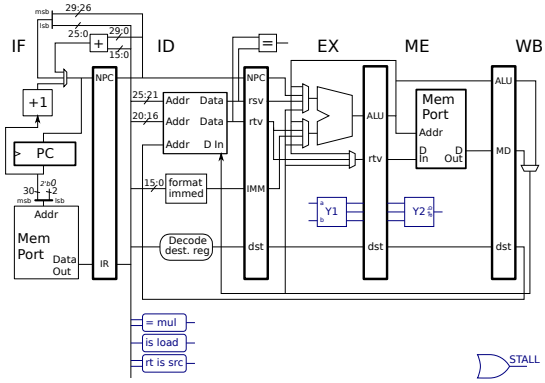
LSU EE 4720 **Homework 2** **Due: 9 March 2020**

Problem 1: The illustration below (and on the next page, there's no need to squint) shows our 5-stage MIPS implementation with some new hardware including: a `Y1` unit in `EX` and a `Y2` unit in `ME`. These are the two stages of a pipelined integer multiplication unit. They are to be used to implement a MIPS32 `mul` instruction (not to be confused with a MIPS-I `mult` instruction). The `mul` instruction executes as you would expect it to, for example `mul r1, r2, r3` writes `r1` with the product of `r2` and `r3`. Because of the need to reduce (add together) all of the partial products, the multiplication hardware spans two stages, in contrast to an integer add which is one in one stage (in the ALU of course). *Note: The* `mult` *instruction was the subject of 2013 Homework 4.*

Here is how `mul` should execute:

```
# Cycle          0  1  2  3  4  5  6  7  8
 add R1, r2, r3  IF ID EX ME WB
 mul r4, R1, r5     IF ID Y1 Y2 WB
 mul R6, r7, R1        IF ID Y1 Y2 WB
 sub r8, R6, r9           IF ID -> EX ME WB
# Cycle          0  1  2  3  4  5  6  7  8
```



First of all, notice that there is no problem overlapping the two multiplies. Also notice that there is no problem bypassing a value to the source of a multiply.

(*a*) Add datapath hardware so that the multiply can execute as shown above.

- Assume that the `Y1` and `Y2` units each have about two multiplexor delays of slack. (Meaning if the path into the inputs of `Y1` or out of the output of `Y2` passes through more than two multiplexors the clock period would have to be increased, and we don't want that.)
- Pay attention to cost. Assume that the cost of one pipeline latch bit is the same as two multiplexor bits. Make other reasonable cost assumptions.
- Do not lengthen the critical path.
- Make sure that the code fragment above will execute as shown.
- Don't break other instructions.

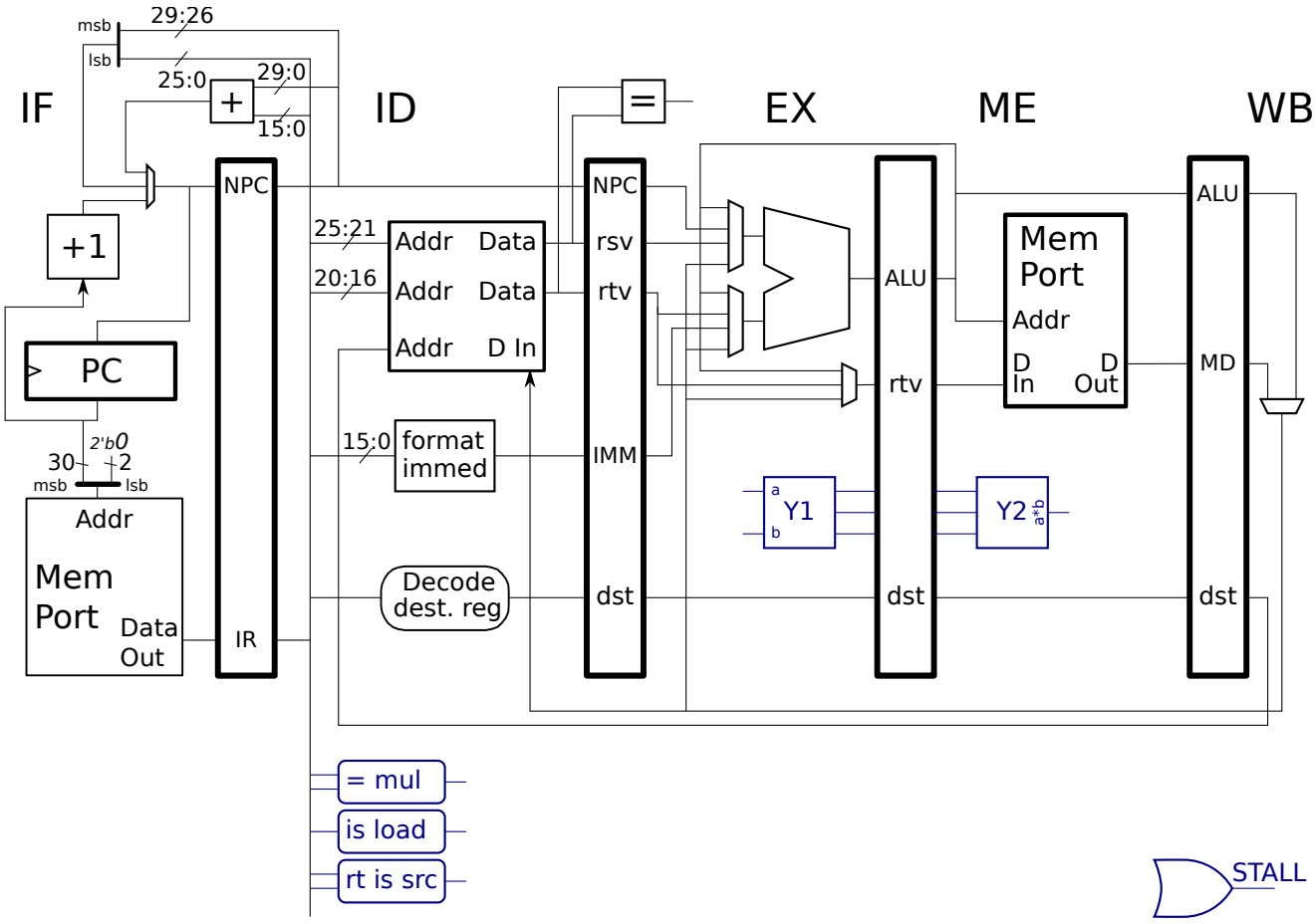
(*b*) Add control logic for the existing `WB`-stage multiplexor and for any new multiplexors you might have added. *Hint: This problem is easy, especially if you use two-input muxen.*

- Use a pipeline execution diagram (such as the one above) to make sure that the value computed for a multiplexor select signal is the correct value when it is used, perhaps several stages later.

(*c*) At the lower-right is a big OR gate, its output is labeled `STALL`. Add an input to that OR gate which will be one when an instruction must stall due to a dependency with a `mul`. The `sub` from the execution above suffers such a stall.

Use next page for solution.

(Not interesting enough? There is another problem on the next page!) Use this page for the solution or download illustration Inkscape SVG source from `https://www.ece.lsu.edu/ee4720/2020/hw02-p1.svg` and use that one way or another to prepare a solution.



Problem 2: Though two stages (`Y1` and `Y2`) may be necessary to compute the product of arbitrary 32-bit signed integers, there are special cases that can be computed in less time, for example when either operand is zero or one.

If the `Y` units compute the product then it doesn't matter what operation the ALU is set to, but to handle special case(s) suppose that the control logic set the ALU operation to bitwise AND when decoding a `mul` instruction. In that case the output of the ALU would be correct for some multiplication operations and so the product would be ready in time to bypass to the next instruction. Add control logic to detect such situations and suppress the stall when present. Don't design the logic to set the ALU operation itself, we'll leave that to the Magic Cloud [tm].