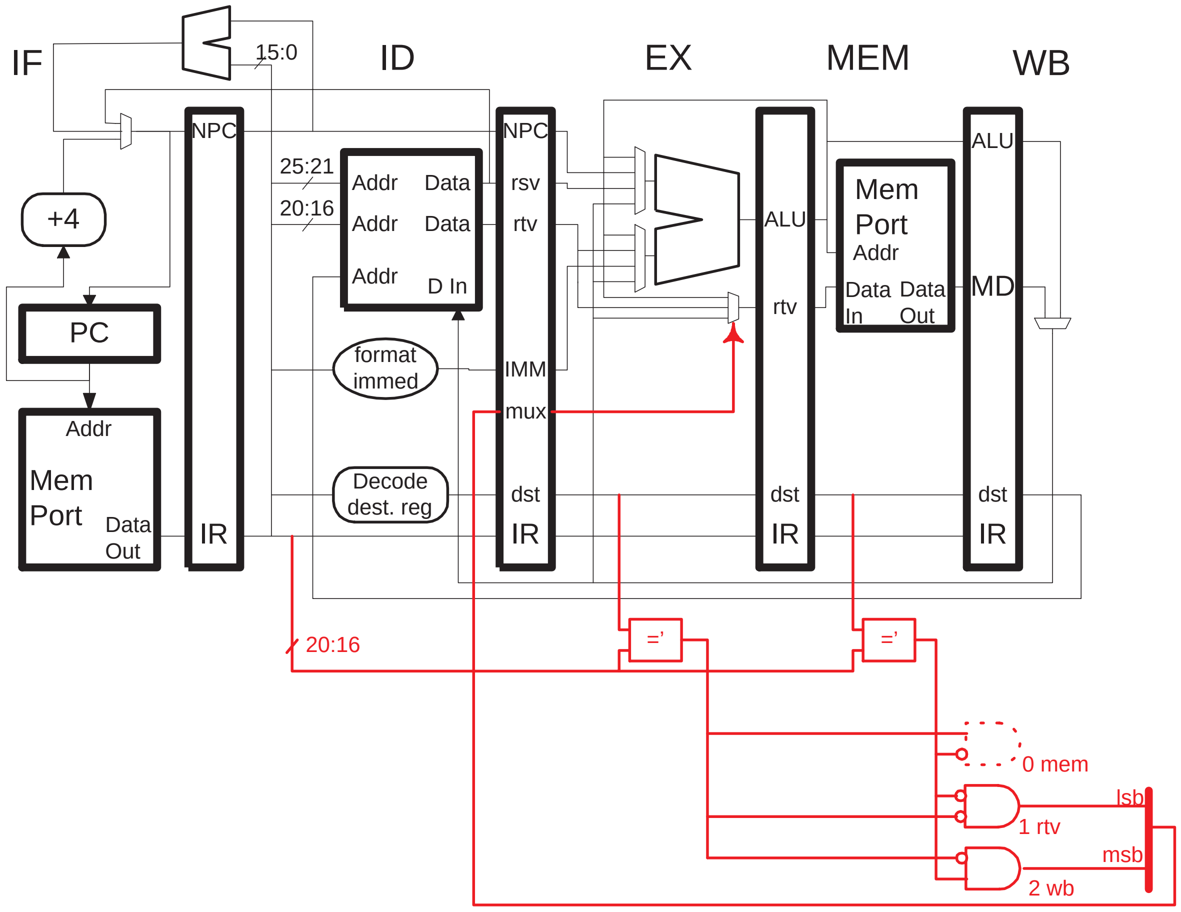
LSU EE 4720 **Homework 4** Solution **Due: 14 November 2003**

Problem 1: Design the control logic for the store value multiplexor (the one that writes pipeline latch `ex_mem_rtv`). The control logic must be in the ID stage. *Hint: This is a fairly easy problem.*

Changes for solution shown in red. Notice that the control logic doesn't check whether the instruction is a store. If it's not a store the store value multiplexor is not used and so it doesn't matter how it's set.



Problem 2: One problem with a post-increment load is storing the incremented base register value into a register file with one write port. Suppose a post-increment, register-indirect load were added to MIPS and implemented in the pipeline on the next page. This post-increment load does not use an offset, instead the effective address is just the contents of the `rs` register.

One option for storing the incremented base register value is to stall the following instruction and write back the value when the bubble reaches WB. We would like to avoid stalls if we have to, so for this problem design hardware that will use the WB stage of the instruction before [sic] or after the post-increment load if one of those instructions does not perform a writeback. For example:

```
bneq $s0, $s1, SKIP (Not taken)
lw $t1, ($t2)+
j TARG

add $s3, $s1, $s2
lw $t1, ($t2)+
sub $s4, $s5, $s6
```

The first post-increment load could writeback when either the `bneq` or the `j` were in the WB stage since neither performs writeback. The second post-increment load would have to insert a stall.

(*a*) Show the hardware needed to implement the post-increment load in this way.

- Remember that this load does not have an offset.
- Use a $\boxed{\text{=PIL}}$ box to identify post-increment loads (input is opcode, output is `1` if it is a post-increment load, `0` otherwise).
- A stall signal is available in each stage; if the signal is asserted the instruction in that and preceding stages will stall and a `nop` instruction will move into the next stage (for each cycle hold is asserted).
- Show any new paths added for the incremented value, perhaps to the register file write port (which still has one write port).
- Add any new paths needed to get the correct register number to the register file write port.
- Ignore bypassing of the incremented address to other instructions.
- Show the added control logic, which does **not** have to be in the ID stage. (In fact it would be difficult to put all of control logic for this instruction in the ID stage.)
- **Last but not least,** a design goal is low cost, so add as little hardware as necessary to implement the instruction.

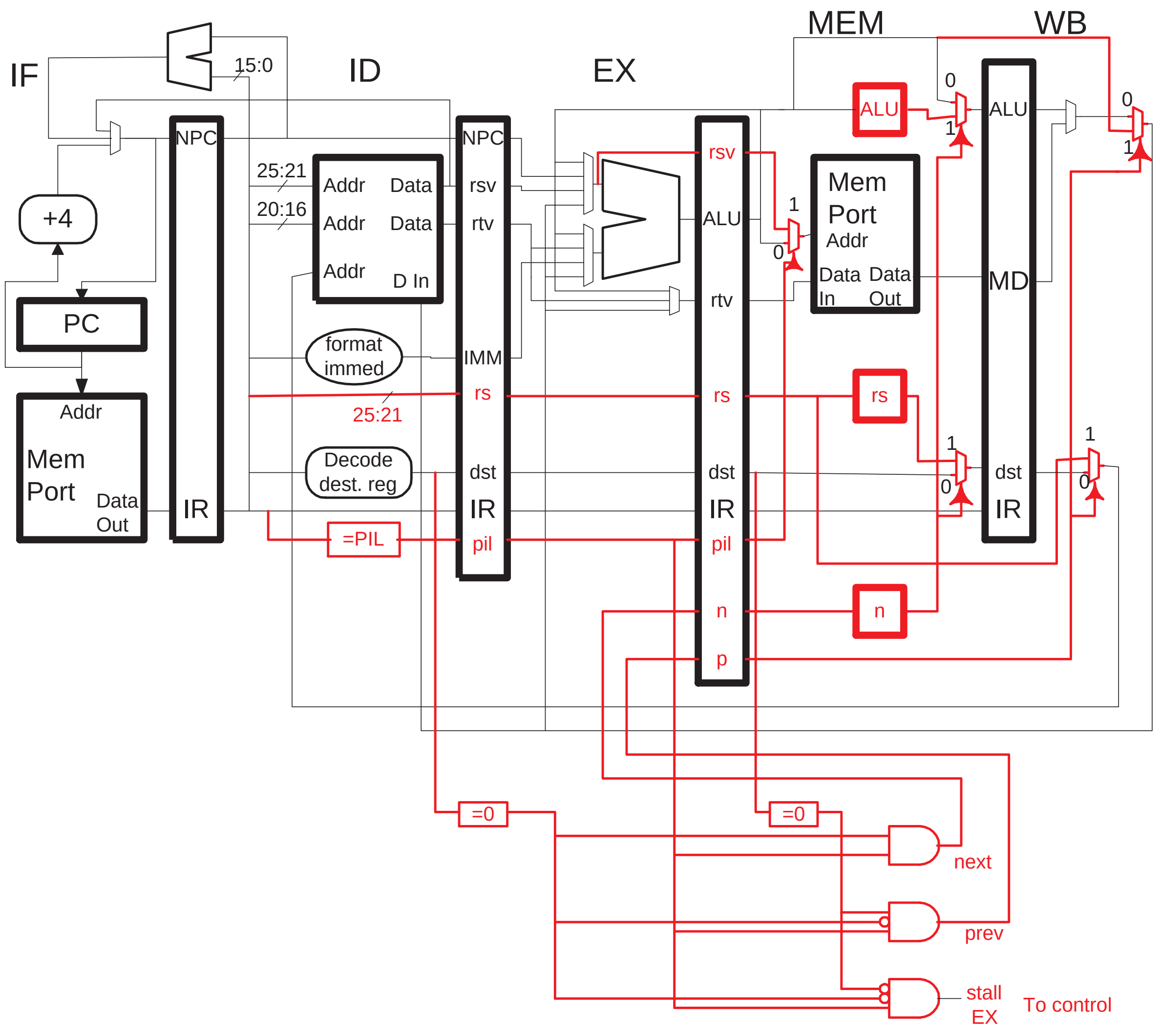
(*b*) If you're like most people, you didn't worry about precise exceptions when solving the previous part. Explain how the need for precise exceptions can complicate the design.

If the post-increment raises an exception in Mem then it might be too late to prevent writing back the incremented address if it's using the previous instruction. Another case is the post-increment load writing back using the next instruction. If the next instruction raises an exception one would have to make sure that despite being squashed it still wrote back the incremented address.

Changes shown in red. In this solution the ALU computes the incremented address, a multiplexor is added so the unincremented address can be sent to the memory's address input. An alternative solution would have an adder dedicated to incrementing the address; with such an adder one would not need the multiplexor at the memory address input.

The control logic, in EX, generates three signals, **next**, **prev**, and **stall EX**. Signals **next** and **prev** are put in pipeline latches, **stall EX** goes to control logic (not shown) to stall EX, ID, and IF. The **prev** signal sets the pipeline to write back the incremented address in the WB of next previous instruction (which is in the next stage), this is done by having the incremented address and the rs field skip ahead one stage. The **next** signal has the rs field and incremented address hold back one cycle by routing them through an extra set of registers, ALU, rs, and n in the MEM stage.

For lower cost, the ALU register added to the Mem stage can be eliminated. Instead, one could use an enable signal to hold the value in the Mem/WB.ALU latch.



Problem 3: Answer these questions about interrupts in the PowerPC, as described in the PowerPC Programming Environments Manual, linked to `http://www.ece.lsu.edu/ee4720/reference.html`.

(*a*) Listed below are the three types of interrupts using the terminology presented in class. What are the equivalent terms used for the PowerPC.

- Hardware Interrupt

  Asynchronous Exception

- Exception

  Synchronous Exception

- Trap

  Trap?

(*b*) In which register is the return address saved?

It is saved in `SSR0` (save/restore register 0).