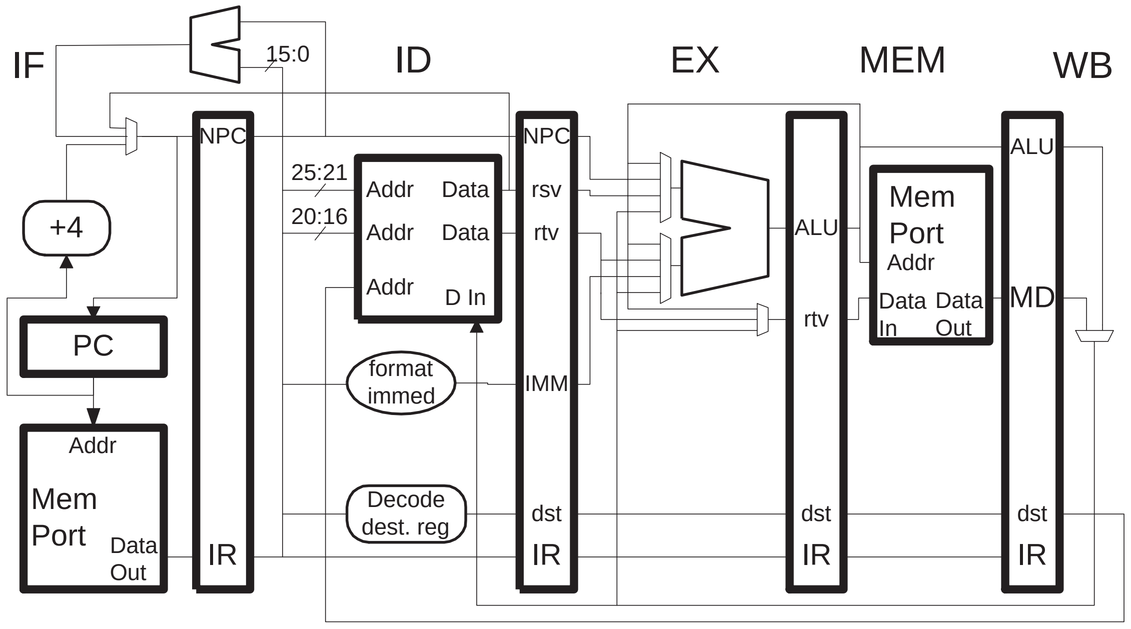
LSU EE 4720 **Homework 4** **Due: 14 November 2003**

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.*



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:

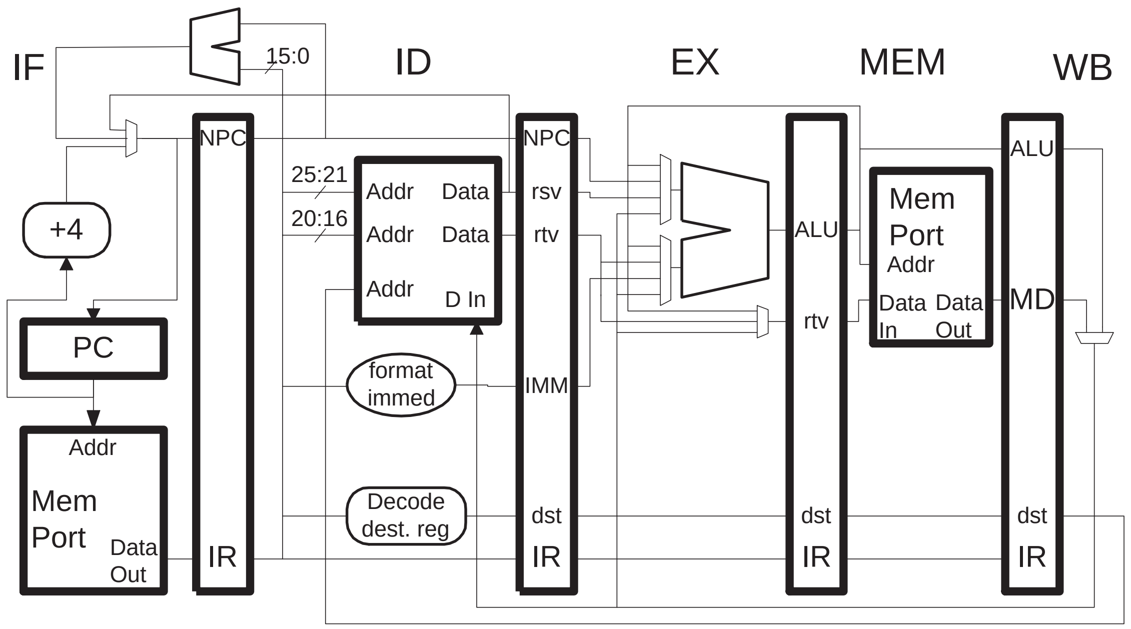
```
bneq $s0, $s1, SKIP (Not taken)
lw $t1, ($t2)+
j TARG

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
15:0
ID
EX
MEM
WB
NPC
NPC
25:21
Addr
Data
rsv
20:16
Addr
Data
rtv
+4
Addr
D In
ALU
rtv
ALU
Mem
Port
Addr
Data
In
Data
Out
MD
PC
format
immed
IMM
Addr
Mem
Port
Data
Out
IR
Decode
dest. reg
dst
IR
dst
IR
dst
IR

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
- Exception
- Trap

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