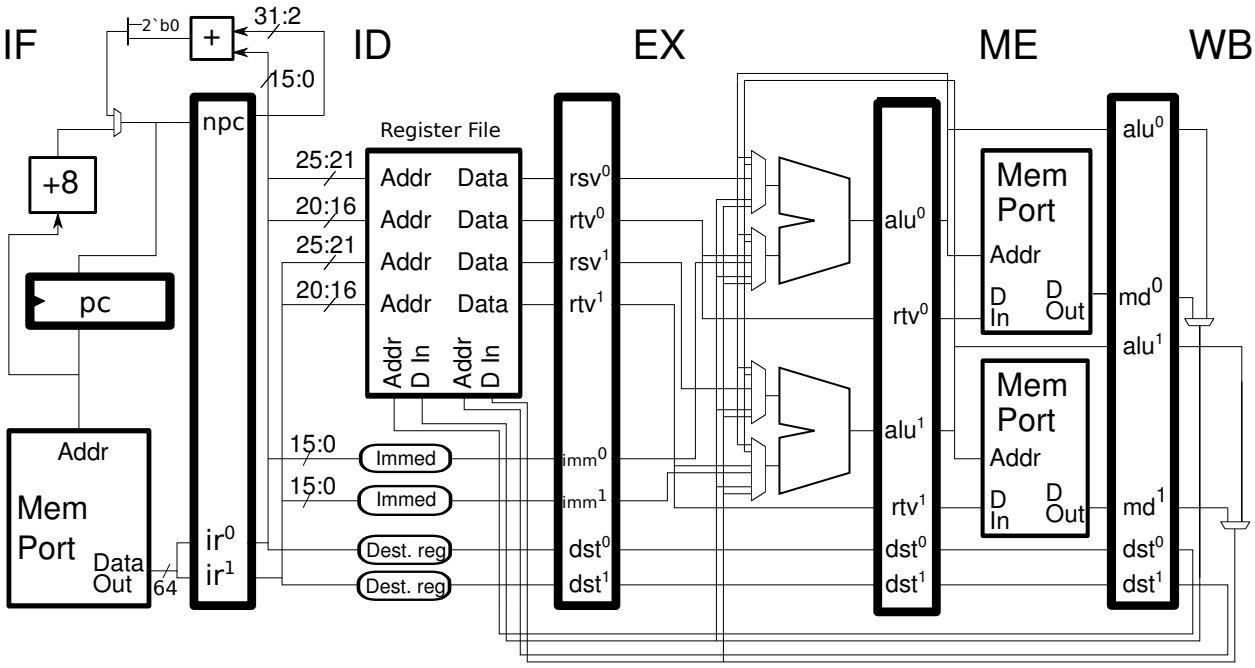
LSU EE 4720 **Homework 6** Solution **Due: 12 April 2013**

SVG and EPS versions of the superscalar processor illustration are available at `http://www.ece.lsu.edu/ee4720/2013/mpipei3ss.svg` *and* `http://www.ece.lsu.edu/ee4720/2013/mpipei3ss.eps`*, respectively. Inkscape can be used to edit the SVG version.*

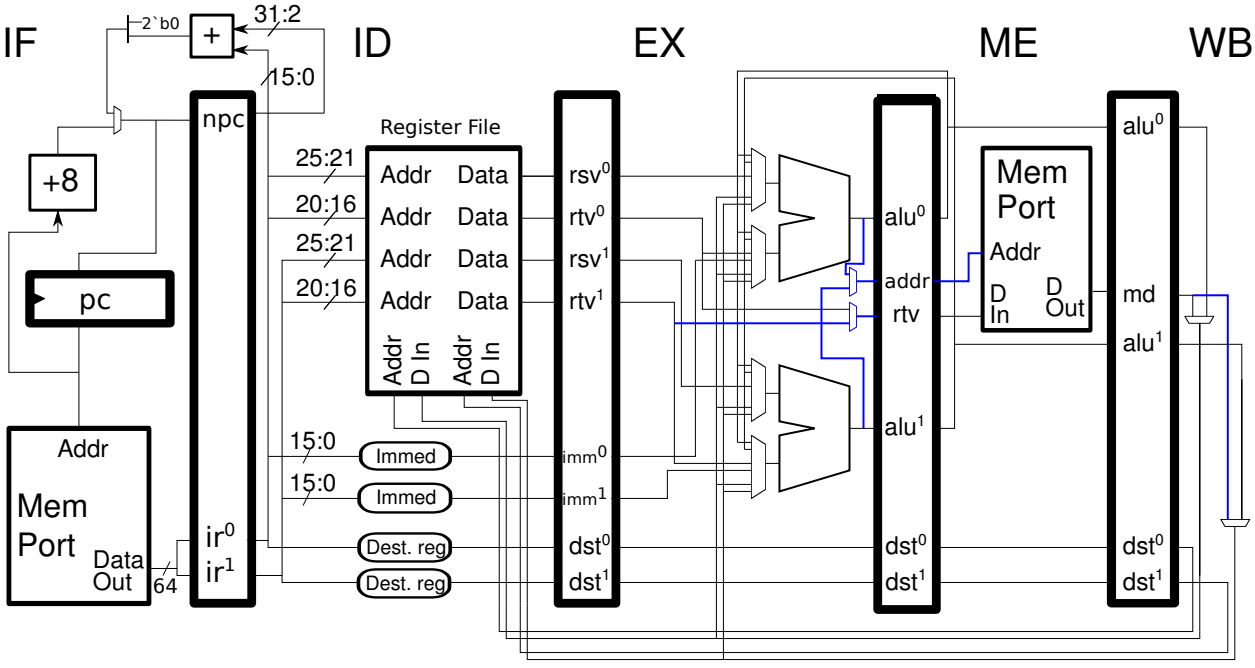
Problem 1: The two-way superscalar implementation below has two memory ports in the `ME` stage, and so it can sustain an execution of 2 IPC on code containing only load and store instructions. Since for many types of programs loads and stores are rarely so dense and because memory ports are costly, it is better to make a 2-way processor with just one memory port in the `ME` stage.

Modify the implementation below so that it has just one memory port in the `ME` stage. It should still be possible to execute arbitrary MIPS programs, albeit more slowly.

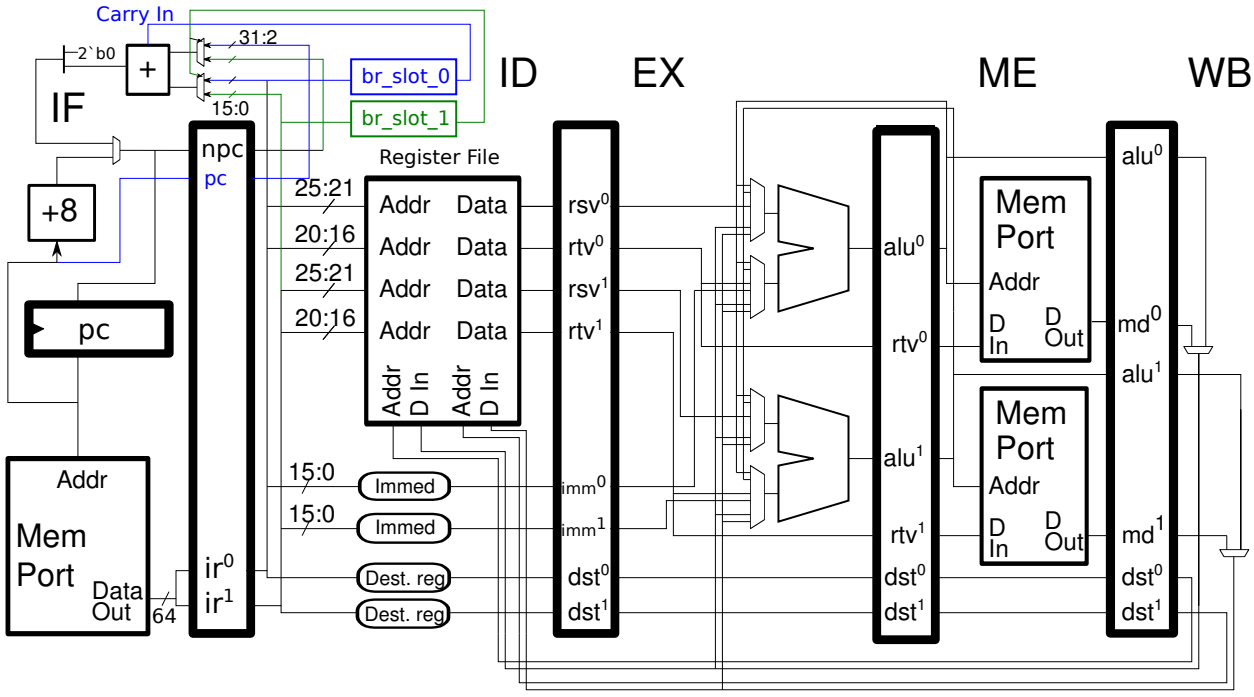
Solution on next page.



Solution appears below, original design above. One of the memory ports was removed, and multiplexors were used to provide paths from either slot to the Addr and D In inputs to the memory ports, these are shown in blue. These are placed in the EX stage (rather than ME) under the assumption that the memory port takes the most time and so anything added between its connections and the pipeline latches would lower the clock frequency. Also note that the output of the memory port connects to the multiplexor for each slot.



Problem 2: The datapath hardware for resolving branches in the 2-way superscalar MIPS implementation below is incomplete: it does not show branch target computation for the instruction in Slot 1 (it is shown for Slot 0). (The hardware for determining branch conditions is also not shown, but that's not part of this problem.) The `IF`-stage memory port can retrieve any 4-byte aligned address. (That is, it does not have the stricter 8-byte alignment that is assumed by default for 2-way superscalar processors presented in class.)



(*a*) Add hardware so that the correct branch target is computed for branches in either slot. The following signals are available: `br_slot_0`, which is 1 if the instruction in Slot 0 (ir^0) is a branch; `br_slot_1`, which is 1 if the instruction in Slot 1 is a branch. Assume that there will never be a branch in both Slot 0 and Slot 1.

Design the hardware for low cost. *Hint: Adder carry-in inputs can come in handy.* The goal of this part is to generate the correct target address, the next part concerns what is done with it.

Solution appears above in blue and green. Multiplexors have been placed before both adder inputs, and the `br_slot_1` signal is used as a control input. One mux provides the corresponding displacement value. The other provides either `pc` or `npc`. For the branch in slot 0 we need to compute `pc + immed0 + 4`, so the upper input to the mux is `pc` and the carry in bit is set to 1. (Note that the new `ID.pc` latch holds the address of the instruction in slot 0.) For the branch in slot 1 we need to compute `pc_1 + immed1 + 4`, where `pc_1` is the address of the instruction in slot 1, but `pc_1` is not available anywhere. However, `npc` is `pc_0+8 = pc_1 + 4`. So for a branch in slot 1 we compute `npc + immed1` (in part by setting the carry in to 0).

(*b*) Add datapath so that the branch target address can be delivered to the PC at the correct time, whether the branch is in Slot 0 or Slot 1. (Earlier in the semester branch delay slots were given as an example of an ISA feature that worked well for the first implementations but that would become a burden in future ones. Welcome to the future.)

No datapath needs to be added. The question should have asked for control logic to identify instructions to squash.