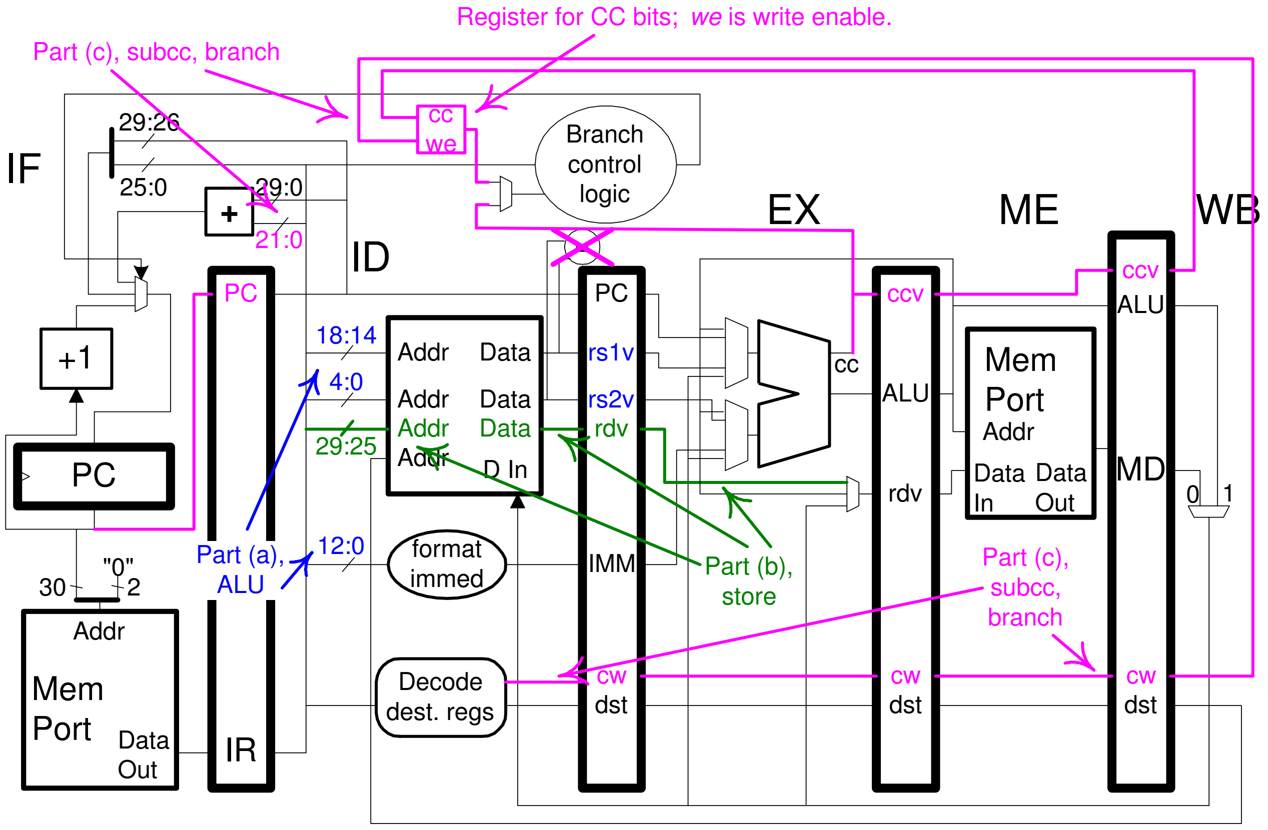
LSU EE 4720 **Homework 3** Solution **Due: 20 October 2006**

Problem 1: Show the changes to the MIPS implementation below needed to implement the SPARC V8 instructions shown in the sub-problems. (See the SPARC Architecture Manual linked to the course references page for a description of SPARC instructions.) Do not show control logic changes or additions. For this problem assume that SPARC has 32 general-purpose registers, just like MIPS. (In reality there are $16n$, $n \geq 4$ general-purpose registers organized into windows. An integer instruction sees only 32 of these but using `save` and `restore` instructions a program can replace the values of 16 of them, the feature is intended for procedure calls and returns. To satisfy curiosity, see the description of register windows in the ISA manual.)



For solution can use larger version on next page.

(*a*) Show the changes for the following instructions. The only changes needed for these are to bit ranges in the ID stage.

```
add %g1, %g2, %g3
sub %g4, 5, %g6
```

Changes shown in blue in the diagram. The instruction bits used for the two existing read ports on the ID-stage register file changed to 18:14 and 4:0, so that the SPARC `rs1` and `rs2` registers would be retrieved. The bits in to the ID-stage format-immed logic changed to 12:0, reflecting the SPARC immediate field.

(*b*) Show the changes needed for the store instruction below. This will require more than changing bit ranges.

```
st %g3, [%g1+%g2]
```

Changes shown in green in the diagram. The store instruction has three source operands so a third read port added to the register file. In EX, the MUX leading to the ME-stage Data In port now gets its input from rdv (the new register read port value).

(*c*) Show the changes needed to implement the instructions below. The alert student will have noticed the ALU has a new output labeled `cc`. That output has condition code values taken from the result of the ALU operation.

- Don't forget the changes needed for the branch target.
- The changes should work correctly whether or not the branch immediately follows the CC instruction.
- Cross out the comparison unit if it's no longer needed.

```
subcc %g1, %g2, %g3
bge TARG
```

Changes shown in purple in the diagram. The cc instructions write the condition-code register, which is like any other register and so is placed in the ID stage. The CC value is computed by the cc output of the ALU, and that is carried along the pipeline in new CC pipeline latches to the WB stage where a new CC register is written. That new register has a write-enable (we) input so that only cc instructions (such as `subcc`) will write it. The output of the cc register connects to the branch control logic, a cc value is bypassed from the EX stage so that a branch immediately after a cc instruction (as in the example above) doesn't have to stall.

SPARC branch instruction targets are computed as a displacement from PC rather than NPC, so IF/ID latch changed. The ID-stage branch target adder lower input changed to reflect the position and size of the displacement field in SPARC instructions, bits 21:0.

Grading Notes: Some submitted solutions have the cc register bits compared to the `comp` field in the branch instruction. That won't work because the `comp` field does not specify exactly what the cc bits should be. For example, `be` (branch equal to zero) just checks if the Z bit is set.

Some submitted solutions show the CC register being written in the instruction's EX stage. That won't work because the instruction may be squashed after EX (and so it would not easily be possible to recover the old cc value). Given what's been covered so far it may seem like a squash won't happen to an instruction that reaches EX, but that will change when we cover exceptions.