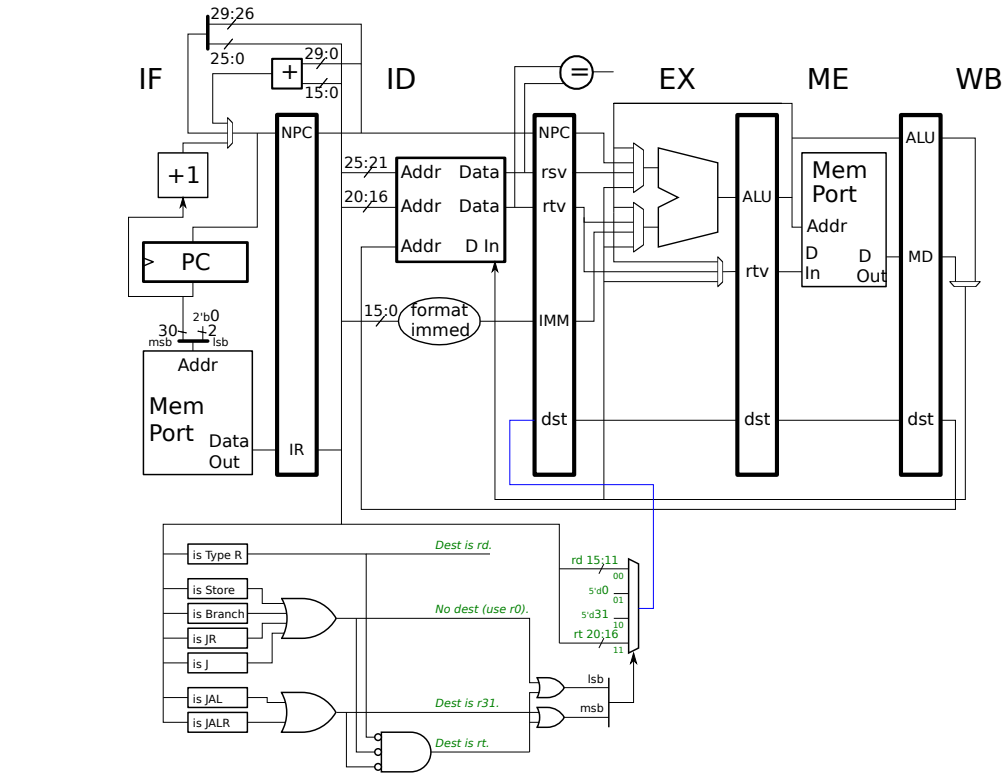
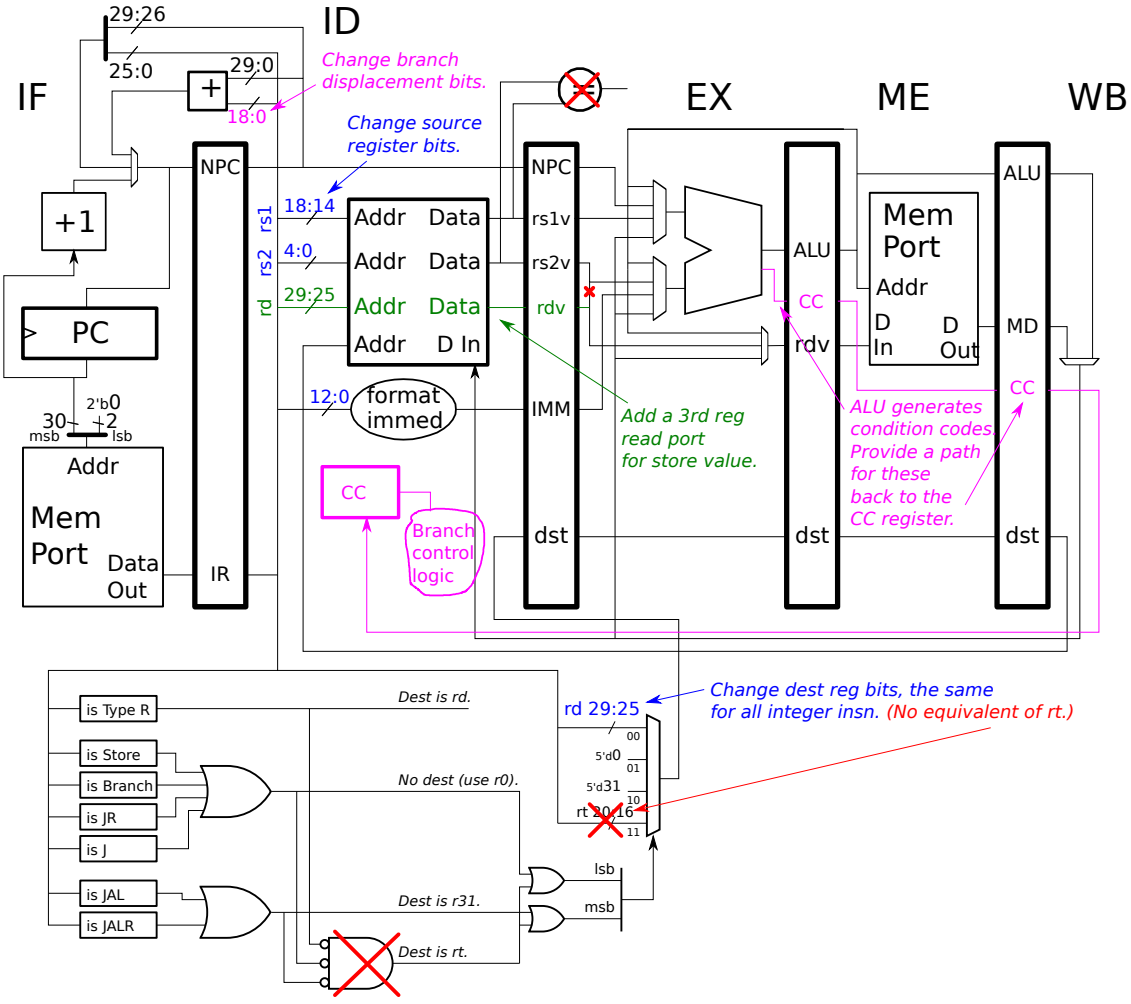
LSU EE 4720 **Homework 3** Solution **Due: 28 March 2016**

Problem 1: Illustrated below is our MIPS implementation with some control logic shown. Modify the implementation so that it can execute the SPARC v8 instructions as described below. In your solution ignore register windows, assume that SPARC uses an ordinary 32-register general-purpose register file.

Details of the SPARC ISA (which includes later versions) can be found in `http://www.ece.lsu.edu/ee4720/doc/JPS1-R1.0.4-Common-pub.pdf`. An Inkscape SVG version of the illustration below can be found at `http://www.ece.lsu.edu/ee4720/2016/mpipei3b.svg`.

Solution on next page.


IF
ID
EX
ME
WB
29:26
25:0
29:0
15:0
+1
PC
2'b0
30
+2
msb
lsb
Addr
Mem
Port
Data
Out
IR
NPC
25:21
20:16
Addr Data
Addr Data
Addr D In
15:0
format
immed
rsv
rtv
IMM
dst
ALU
rtv
dst
Mem
Port
Addr
D
In
D
Out
ALU
MD
dst
is Type R
is Store
is Branch
is JR
is J
is JAL
is JALR
Dest is rd.
No dest (use r0).
Dest is r31.
Dest is rt.
rd 15:11
5'd0
5'd31
rt 20:16
lsb
msb


ID
IF
EX
ME
WB
29:26
25:0
29:0
18:0
Change branch
displacement bits.
Change source
register bits.
+1
PC
2'b0
30
+2
msb
lsb
Addr
Mem
Port
Data
Out
IR
NPC
rs1
rs2
rd
18:14
4:0
29:25
Addr Data
Addr Data
Addr Data
Addr D In
12:0
format
immed
CC
Branch
control
logic
rs1v
rs2v
rdv
IMM
dst
Add a 3rd reg
read port
for store value.
ALU
CC
rdv
dst
Mem
Port
Addr
D
In
D
Out
ALU
MD
CC
dst
ALU generates
condition codes.
Provide a path
for these
back to the
CC register.
is Type R
is Store
is Branch
is JR
is J
is JAL
is JALR
Dest is rd.
No dest (use r0).
Dest is r31.
Dest is rt.
rd 29:25
5'd0
5'd31
rt 20:16
lsb
msb
Change dest reg bits, the same
for all integer insn. (No equivalent of rt.)

(*a*) Modify the implementation for format 3 arithmetic instructions. Use `add` as an example. Show changes in the bits used: to index the register file, to format the immediate, and to generate the writeback register number, `dst`.

Solution appears in blue on the previous page. The bits at the input to the register file have been changed, and notice also that the pipeline latches at the register file outputs were renamed, for example, from rsv to rs1v. Since all SPARC integer instructions that write a value, use the `rd` field for that value. For that reason the `dst` mux input for `rt` has been removed. Also note that the immediate unit uses fewer bits. (If the `sethi` instruction were implemented then the number of bits might be expanded.)

(*b*) Modify the implementation for branch instructions. Use `BPcc` as an example. Be sure to make changes for computing the branch target.

Show changes in the hardware to generate the target address. Remove the unneeded MIPS branch comparison hardware and add a CC register.

Solution shown in purple. The branch displacement bits were changed at the input to the ID-stage adder. Also, a CC output was added to the ALU and the value is carried through the pipeline to the ID stage where it's used to write a new CC register. (It goes without saying that bypass paths could be added to the branch control logic.) Also notice that he comparison unit in ID has been removed (shown with a red ex).

(*c*) Modify the implementation for load and store instructions. Use `LDUW` and `STW` as examples.

Show changes in the format immediate unit, and make sure that it can handle both `ADD` and loads and stores.

Changes shown in green. Only the `STW` instruction requires further changes. An instruction like `stw r1, [r2+r3]` has three source registers, and so a third read port has been added to the register file. A path for the retrieved value, `rdv`, is provided to the Mem Port Din and red ex breaks the old path (from what was rtv).

Problem 2: Section 1.3.1 of the SPARC JPS1 lists features of the ISA.

(*a*) Indicate which features are typical RISC features and which features are not.

The typical RISC features are those that facilitate pipelined implementations and make it easy to compile code. They are 32-bit instructions (as opposed to variable-sized instructions), few addressing modes, and triadic register addresses (as opposed to having arithmetic instructions read memory or forcing an arithmetic instruction to use the same register for its first source and destination, or something like that).

(*b*) One feature is "Branch elimination instructions" Provide an example of how such an instruction can be used to eliminate a branch.

The conditional moves are the branch elimination instructions. They write a register if a condition is true. For example, `movg r1, r2` (move greater than) will write `r2` with the value of `r1` only if the ICC register Z (zero) and N (negative) bits are both zero (meaning the last CC instruction result was strictly greater than zero). If the condition is not true `r2` is unchanged.

See page 276 of the SPARC JPS1 for an example.

Note: other ISAs, such as ARM, achieve the same result using predication.