### Modifications:

1. Inhibit “Reserved Instruction Exception”.

In module *m14k\_mpc\_dec*, change:

assign spec\_ri\_e = … … (mpc\_ir\_e[5:1] == 5'b101\_00) ||

For:

assign spec\_ri\_e = … … (mpc\_ir\_e[5:0] == 6'b101\_001) ||

1. Select the SLT result (in which we will include also SEQ result) instead of the ALU result in *\_res\_m\_31\_0\_* multiplexer (module *m14k\_edp\_buf\_misc*):

mvp\_mux2 #(32) \_res\_m\_31\_0\_(res\_m[31:0],mpc\_udislt\_sel\_m, asp\_m, {31'h0, bit0\_m});

This multiplexer is governed by signal *mpc\_udislt\_sel\_m*, which is 1 for a *slt* (*slt\_sel\_m*=1) or *udi* (*mpc\_udisel\_m*=1) instruction and 0 otherwise.

assign mpc\_udislt\_sel\_m = slt\_sel\_m | mpc\_udisel\_m;

Thus, we must change computation of *slt\_sel\_m*. At module *m14k\_mpc\_dec*, add one line to computation of this signal. Change:

assign slt\_sel\_e = (mpc\_ir\_e[31:27] == 5'b001\_01) || // slti, sltiu

special\_e && (mpc\_ir\_e[5:1] == 5'b101\_01) || // slt, sltu

special\_e && (mpc\_ir\_e[5:2] == 4'b110\_0) || // tge, tgeu, tlt, tltu

regimm\_e && (mpc\_ir\_e[20:18] == 3'o2); // tgei, tgeiu, tlti tltiu

For:

assign slt\_sel\_e = (mpc\_ir\_e[31:27] == 5'b001\_01) || // slti, sltiu

special\_e && (mpc\_ir\_e[5:1] == 5'b101\_01) || // slt, sltu

special\_e && (mpc\_ir\_e[5:1] == 5'b101\_00) || // SEQ

special\_e && (mpc\_ir\_e[5:2] == 4'b110\_0) || // tge, tgeu, tlt, tltu

regimm\_e && (mpc\_ir\_e[20:18] == 3'o2); // tgei, tgeiu, tlti tltiu

1. We have different options for computing the equality condition at module m14k\_edp:
   1. The obvious solution is to include a new hardware for computing the equality condition (==) among the two operands.
   2. The SLT instruction is computed by substracting rs-rt in the “Carry Propagate Adder” included in module *m14k\_edp*, called *edp\_add*. We can use this operation for computing the equality condition, by ORing the 32 bits resulting from the substraction. In case the result is 0, both operands are equal, otherwise they are different.
   3. We can reuse a comparator already available in the pipeline. This will probably be the cheapest option, so it is the one that we are going to use:

At module *m14k\_edp*, the following hardware is implemented:

assign edp\_cndeq\_e = acmp\_e == edp\_bbus\_e;

We can reuse this hardware, with some small changes:

* Signal *edp\_bbus\_e* can be used as it is.
* Signal *acmp\_e* requires small changes. It is assigned as:

mvp\_mux2 #(32) \_acmp\_e\_31\_0\_(acmp\_e[31:0],mpc\_cmov\_e, edp\_abus\_e, 32'h0);

In this case, we need *edp\_abus\_e* to be selected. Thus, we will include as control signal a new one which is 1 when we have a *seq* instruction.

mvp\_mux2 #(32) \_acmp\_e\_31\_0\_(acmp\_e[31:0],(mpc\_cmov\_e && ~seq\_instr), edp\_abus\_e, 32'h0); // SEQ

where signal *seq\_instr* is computed at module m14k\_mpc\_dec as follows:

assign seq\_instr = special\_e && (mpc\_ir\_e[5:1] == 5'b101\_00); // SEQ

and is registered to the M-Stage:

mvp\_cregister #(1) \_seq\_instr\_m(seq\_instr\_m,mpc\_run\_ie, gclk, seq\_instr); // SEQ

* This signal must be registered to be used at the M-Stage. We add:

mvp\_cregister #(1) \_edp\_cndeq\_m(edp\_cndeq\_m,mpc\_run\_ie, gclk, edp\_cndeq\_e); // SEQ

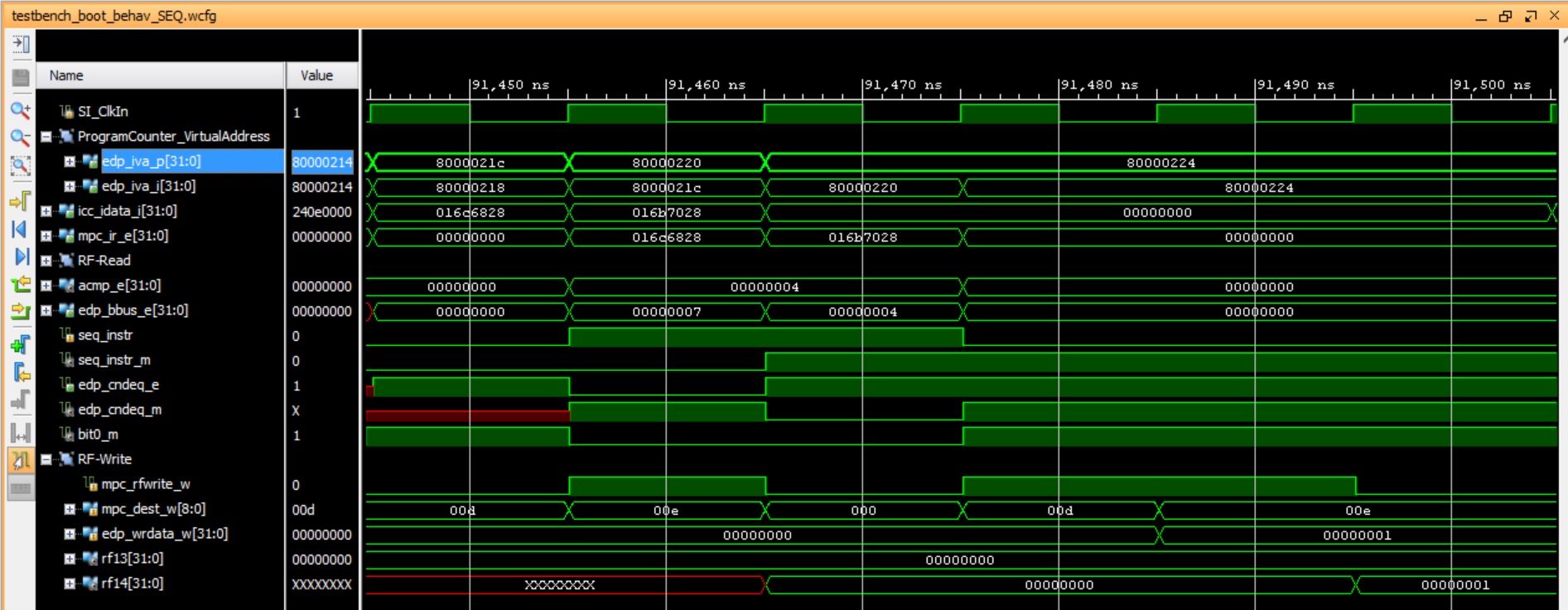
1. At module *m14k\_edp*, we must incorporate the SEQ result to the SLT result (bit0\_m). For that purpose, we must change:

assign bit0\_m = mpc\_signed\_m ? (pro31\_m ^ car31\_m) : ~car31\_m;

For:

assign bit0\_m = seq\_instr\_m ? edp\_cndeq\_m : (mpc\_signed\_m ? (pro31\_m ^ car31\_m) : ~car31\_m); // SEQ

### Simulation:



Observe that, in the fifth cycle rf13($t5)=0 and in the sixth cycle rf14($t6)=1.

### Execution on the Board:

When the program is downloaded on the board, you should see on the 7-seg displays:

* Switches=0 🡪 7-seg displays=$t5, which in our example is 0x0
* Switches=1 🡪 7-seg displays=$t6 which in our example is 0x1
* Any other value for the switches 🡪 7-seg displays=0x0

Then, when you debug the program following the steps stated in the document, you should observe the following:

