**EXPLANATION**:

* + - Let´s first understand the original instructions, MOVZ/MOVN (Figure 3): These instructions are handled like an AND instruction, with some changes:
    - The following signals determine that a MOVZ/MOVN instruction is being executed (module m14k\_mpc\_dec):
      1. ***mpc\_cmov\_e***:
         1. 1 if MOVZ or MOVN instruction.
         2. 0 if any other instruction.
      2. ***cmov\_type\_e***:
         1. 1 if MOVZ.
         2. 0 if MOVN.
      3. ***sel\_logic\_e***: Select logic output.
         1. 1 if MOVZ/MOVN (plus other instructions).
    - Comparison with 0 (module m14k\_edp):
      1. ***edp\_cndeq\_e***: Comparator (used also for other instructions):

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

***edp\_bbus\_e*** is ***Rt***, which is provided from the RF or from a Bypass in case of dependency.

***acmp\_e***, for MOVZ/MOVN instructions (i.e. **mpc\_cmov\_e**=1), is set to *32’h0*:

*mvp\_mux2 #(32) \_acmp\_e\_31\_0\_(acmp\_e[31:0],*

*mpc\_cmov\_e,*

*edp\_abus\_e,*

*32'h0);*

* + - If the condition is met (***edp\_cndeq\_e*** == 1):

The functionality of an AND instruction is used (see example illustrating an AND instruction), with an important change: ***logic\_ain\_e*** (Source-A) = ***logic\_bin\_e*** (Source-B) = ***Rs***. Thus, the result of ***logic\_ain\_e*** & ***logic\_bin\_e*** = ***Rs***.

* + - * 1. Source-A is assigned as in the AND instruction.
        2. Source-B is assigned through a MUX that uses as control signal ***mpc\_alubsrc\_e***, which depends on ***mpc\_cmov\_e***:

1st input (selected when ***mpc\_cmov\_e***==1): ***edp\_abus\_e***

2nd input (selected when logic instruction, etc.): ***bbus\_imm\_e***

* + - If the condition is not met (***edp\_cndeq\_e*** != 1):

***kill\_cmov\_e:*** This signal is set to 1 if the condition for MOVZ/MOVN is not met:

*assign kill\_cmov\_e = mpc\_cmov\_e & (edp\_cndeq\_e ^ cmov\_type\_e) | …*

When ***kill\_cmov\_e*** is 1, the RF write is inhibited:

***! kill\_cmov\_e*** → ***mvd\_e*** → ***pvd\_m*** → ***pvd\_w*** → ***mpc\_rfwrite\_w*** (Register file write enable)

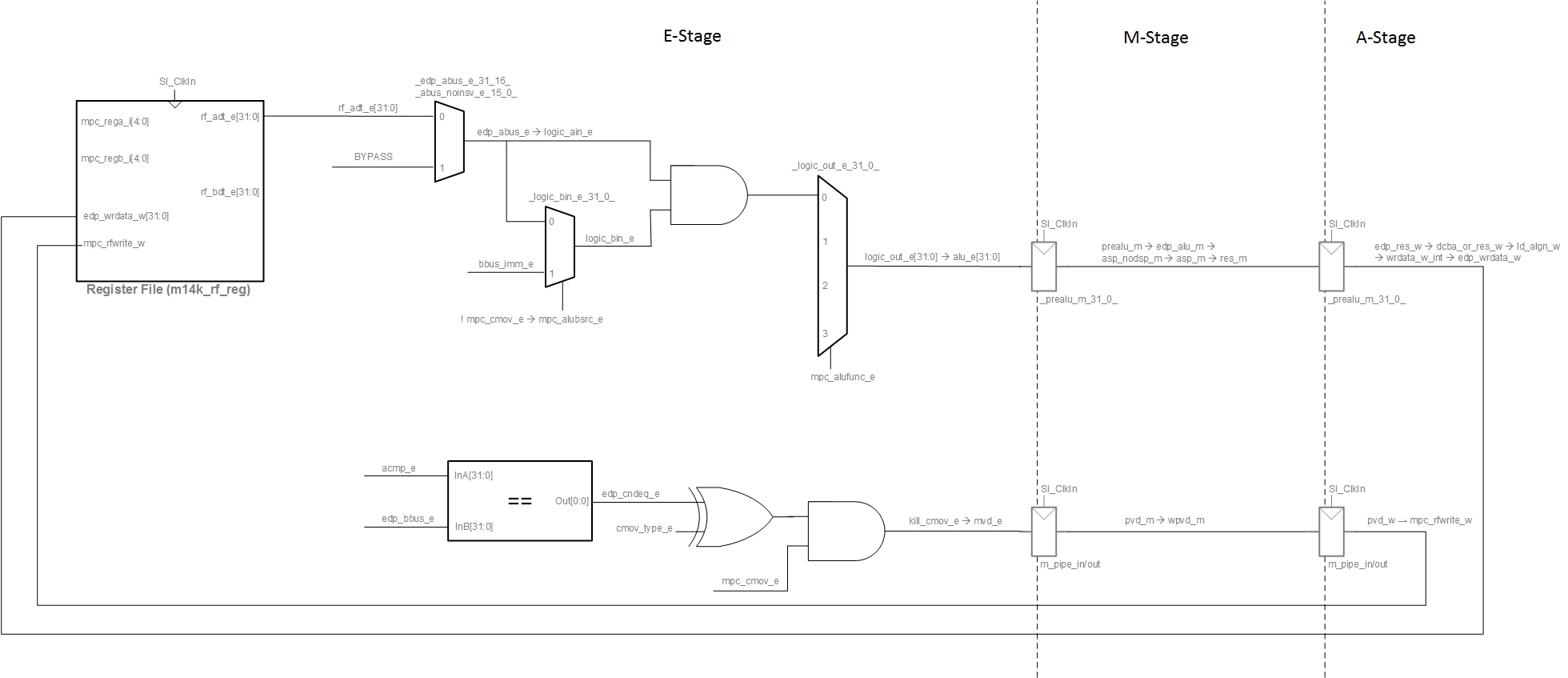


FIGURE 3. Involved structures and signals for MOVZ/MOVN.

* Changes to the original instructions, MOVZ/MOVN, for transforming them into SELEQZ/SELNEZ (Figure 4). All changes related to the new instruction are tagged with comment: *// SELEQZ/SELNEZ* in the soft-core:
  + - Modify ***kill\_cmov\_e*** computation:
      1. In the original instructions, when the condition is not met, ***kill\_cmov\_e*** is set, which inhibits RF write in Stage-W.

**kill\_cmov\_e** *= mpc\_cmov\_e & (edp\_cndeq\_e ^ cmov\_type\_e) | mpc\_movci\_e & ~cp1\_btaken;*

* + - 1. In the new instructions, RF must be written in all cases.

**kill\_cmov\_e** *= mpc\_movci\_e & ~cp1\_btaken;*

* + - Create new signal for distinguishing between condition met / not met:

***cond\_not\_met\_e*** = *mpc\_cmov\_e & (edp\_cndeq\_e ^ cmov\_type\_e)*

This signal is computed at m14k\_mpc\_ctl and passed to m14k\_edp.

* + - A new MUX is included in m14k\_edp for selecting 0s as the Source-B for the AND operation, when ***cond\_not\_met\_e*** is 1. Note that ***cond\_not\_met\_e*** can only be 1 for MOVZ/MOVN instructions.

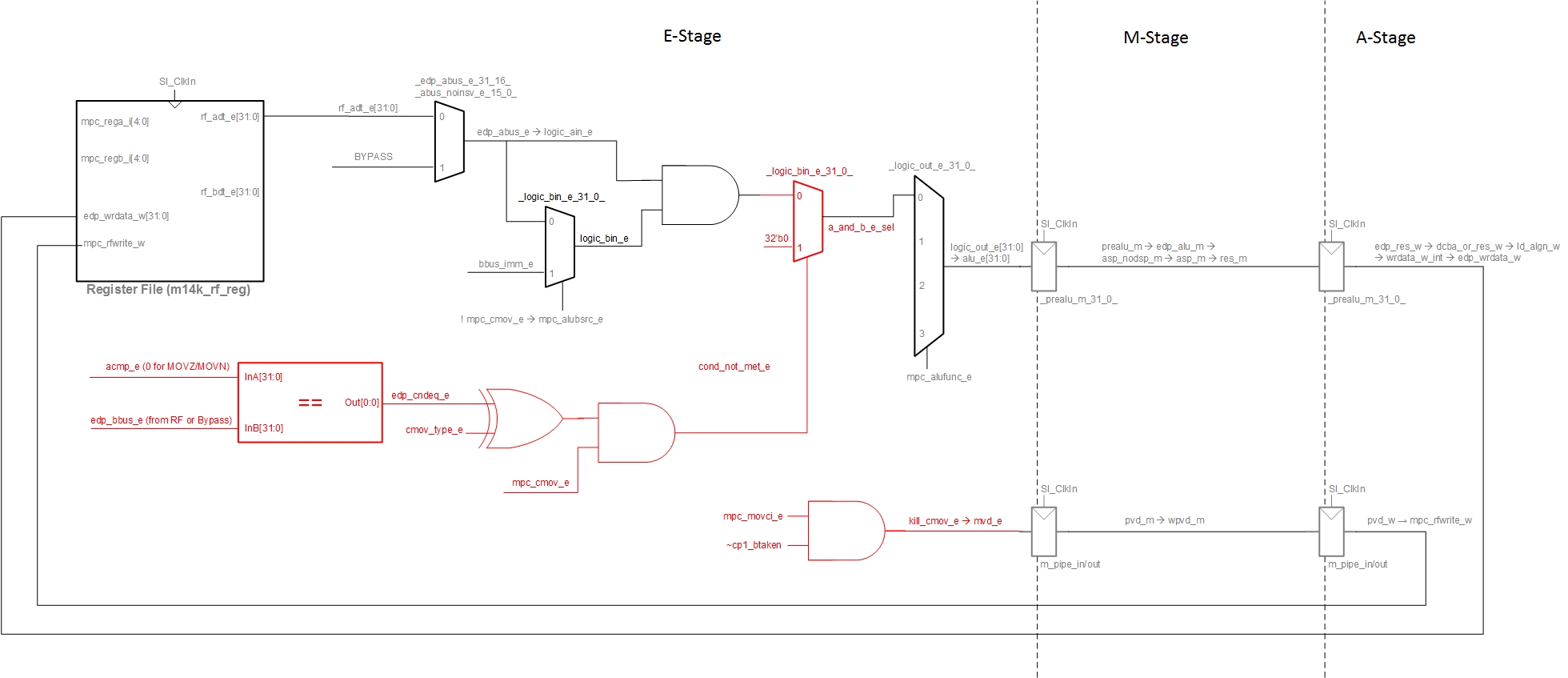


FIGURE 4. Involved structures and signals for SELEQZ/SELNEZ.

**EXAMPLE - SIMULATION**:

**ORIGINAL PROCESSOR**:

**" li $t4, 6;"**

**" li $t3, 0;"**

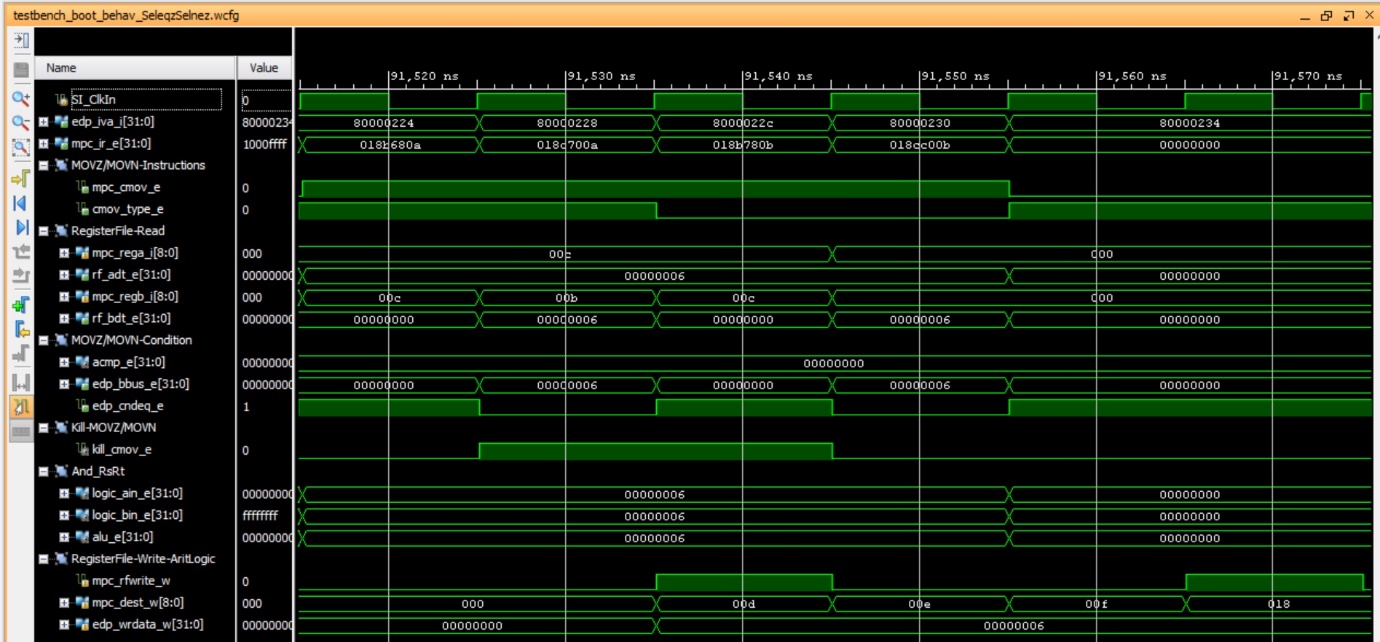
**...**

**" movz $t5,$t4,$t3;"**

**" movz $t6,$t4,$t4;"**

**" movn $t7,$t4,$t3;"**

**" movn $t8,$t4,$t4;"**



Note that in the third cycle, register 0x00d is written to 0x6, and in the sixth cycle, register 0x018 is written to 0x6. Instead, in cycles fourth and fifth the register file is not written.

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

* Switches=0 🡪 7-seg displays=$t5, which in our example is 0x6
* Switches=1 🡪 7-seg displays=$t6 which in our example is 0x1
* Switches=2 🡪 7-seg displays=$t7, which in our example is 0x1
* Switches=3 🡪 7-seg displays=$t8 which in our example is 0x6
* Any other value for the switches 🡪 7-seg displays=0x0

**MODIFIED PROCESSOR**:

**" li $t4, 6;"**

**" li $t3, 0;"**

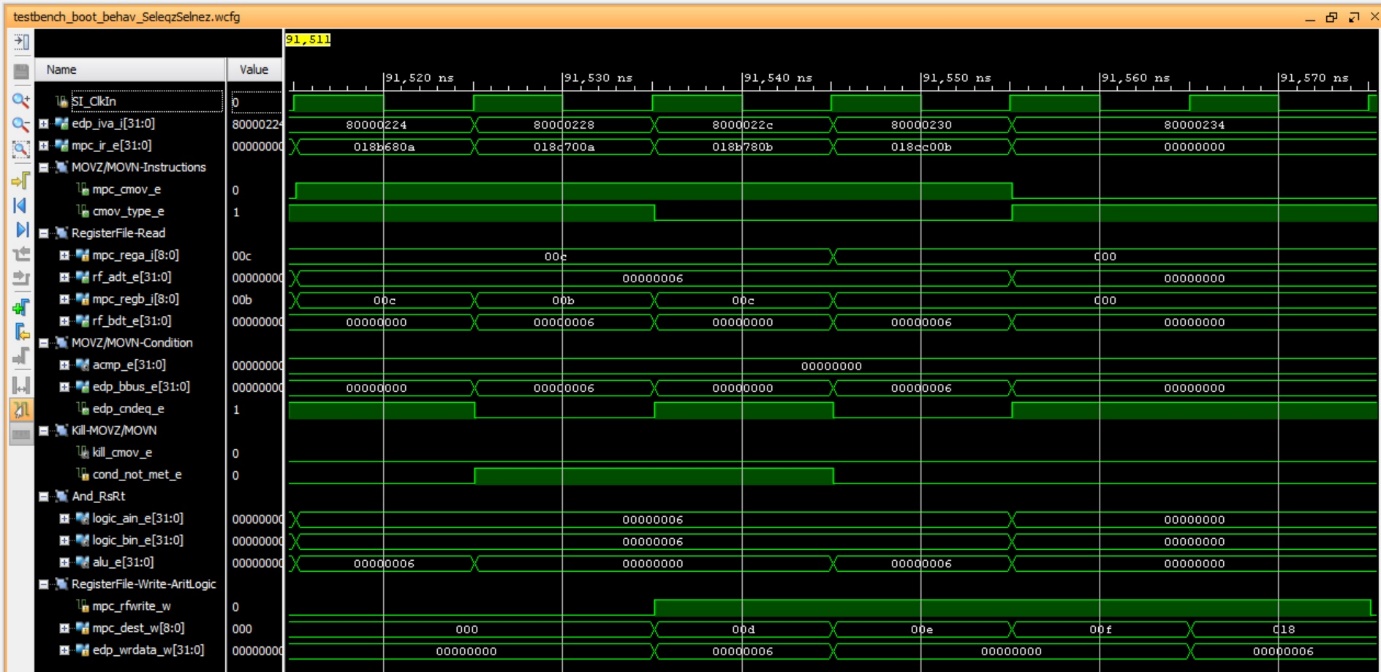
**...**

**" movz $t5,$t4,$t3;"**

**" movz $t6,$t4,$t4;"**

**" movn $t7,$t4,$t3;"**

**" movn $t8,$t4,$t4;"**

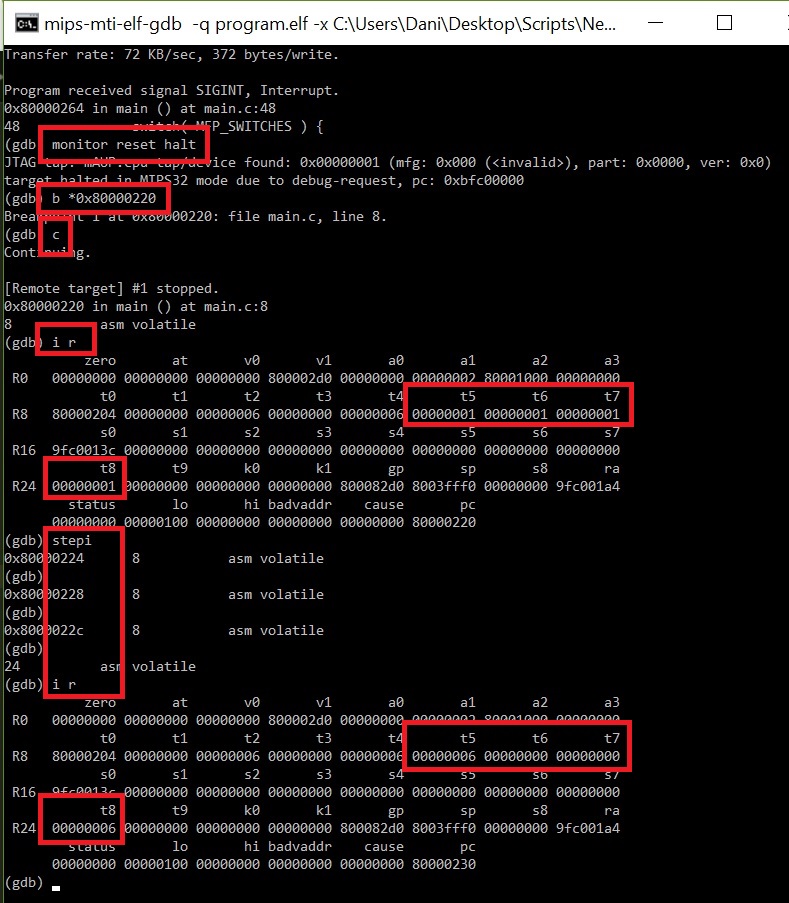


Note that in the third cycle, register 0x00d is written to 0x6, in the fourth cycle, register 0x00e is written to 0x0, in the fifth cycle, register 0x00f is written to 0x0, and in the sixth cycle, register 0x018 is written to 0x6.

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

* Switches=0 🡪 7-seg displays=$t5, which in our example is 0x6
* Switches=1 🡪 7-seg displays=$t6 which in our example is 0x0
* Switches=2 🡪 7-seg displays=$t7, which in our example is 0x0
* Switches=3 🡪 7-seg displays=$t8 which in our example is 0x6
* 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:



**EXTRA FUNCTIONALITY:**

Change the functionality of the new instructions, so that instead of writing a 0 when the condition is not met, ~Rs is written. I.e:

SELEQZ:

*if GPR[rt] = 0 then GPR[rd] ← GPR[rs]*

*else then GPR[rd] ←* **~***GPR[rs]*

SELNEZ:

*if GPR[rt] 0 then GPR[rd] ← GPR[rs]*

*else then GPR[rd] ←* **~***GPR[rs]*

ANSWER:

The only change is in the new Mux, where, instead of 0s:

*mvp\_mux2 #(32) \_and\_output\_31\_0\_(*

*a\_and\_b\_e\_sel[31:0],*

*cond\_not\_met\_e,*

*a\_and\_b\_e,*

*32'b0); // SELEQZ/SELNEZ*

we introduce: ~Rs:

*mvp\_mux2 #(32) \_and\_output\_31\_0\_(*

*a\_and\_b\_e\_sel[31:0],*

*cond\_not\_met\_e,*

*a\_and\_b\_e,*

*~logic\_ain\_e); // SELEQZ/SELNEZ*