EXERCISE 1:

FORWARDING CONTROL SIGNALS:
We´ll do the analysis of the control signals for Source A (control signals for Source B are completely analogous):

1. Signal ***mpc_aselres_e*** controls one of the multiplexers shown in Figure 6. It decides if Source A comes from the RF (***mpc_aselres_e***=0) or Forwarding (***mpc_aselres_e***=1). This signal is computed as the OR of two signals:

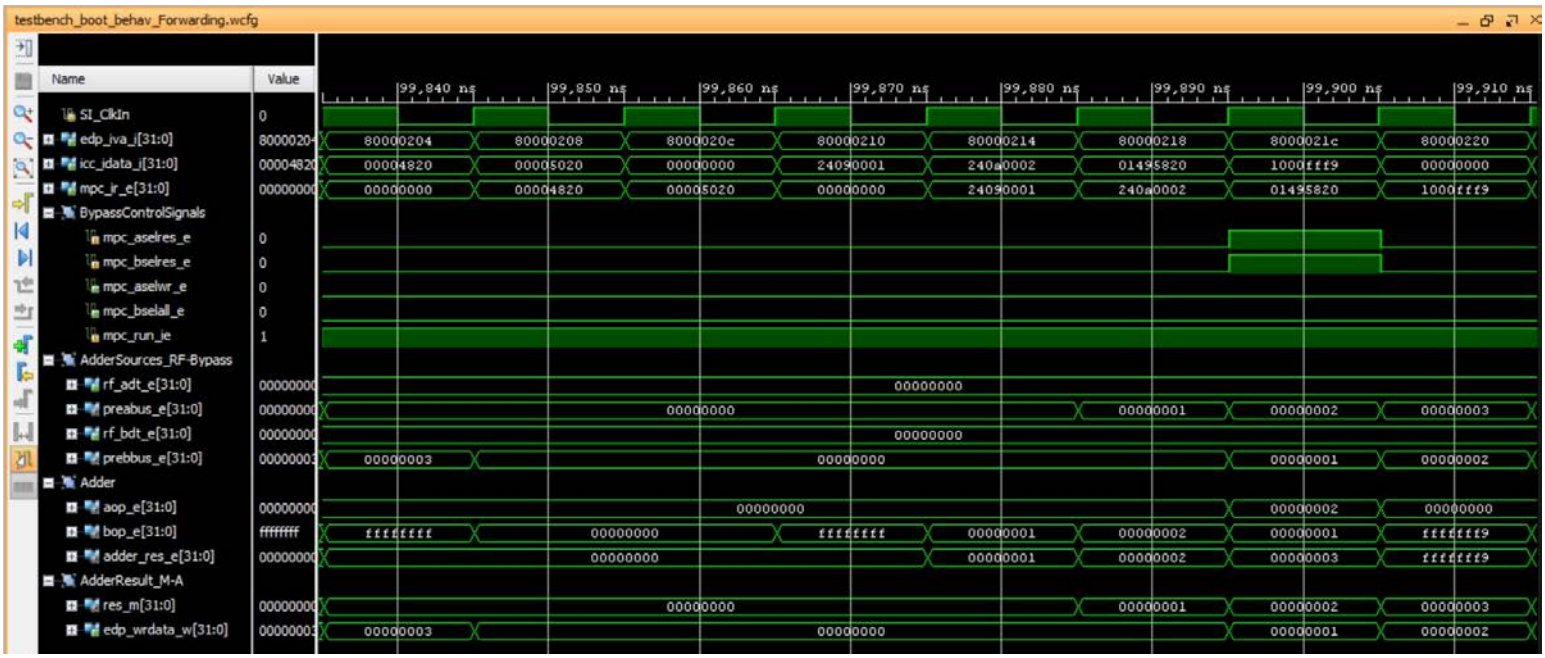
```
assign mpc_aselres_e =    (apsall_e | apswr_e) ^ cpz_rslip_e
```

   a. ***apsall_e***=1 when data must be forwarded from the M-Stage. Computation of this signal is quite complex, but we can simplify it to the AND operation of 2 conditions:
      i. The instruction at the M-Stage will write to the RF (***vd_m***=1): This signal comes from the Decoding of the instruction at the E-Stage, where ***vd_e*** is computed.
      ii. The destination register of the instruction at the M-Stage is the same as Source Operand A of the instruction at the E-Stage (***src_a_e*** == ***dest_m[4:0]***).
   b. ***apswr_e***==1 when data must be forwarded from the A-Stage. Again, computation of this signal is quite complex, but we can simplify it to the AND operation of 2 conditions:
      i. The instruction at the A-Stage will write to the RF (***vd_w***=1).
      ii. The destination register of the instruction at the A-Stage is the same as Source Operand A of the instruction at the E-Stage (***src_a_e*** == ***mpc_dest_w[4:0]***).
2. Signal ***mpc_aselwr_e*** controls the other multiplexers shown in Figure 6 related with Source A. It decides if the forwarding data must come from the M-Stage (***mpc_aselwr_e***=0) or from the A-Stage (***mpc_aselwr_e***=1). Simplifying, this is computed as follows:

```
assign mpc_aselwr_e =     apswr_e & ~apsall_e …
```

   Signals ***apswr_e*** and ***apsall_e*** are explained in the previous ítem. Note that the forwarding from the M-Stage has a higher priority than the forwarding from the A-Stage.

SIMULATION:



In the seventh cycle the add $t3, **$t2**, **$t1** instruction is executed. The first operand ($t2) is provided through forwarding (*mpc_aselres_a*=1) from the M-Stage (*mpc_aselwr_e*=0), and the second operand ($t1) is also provided through forwarding (*mpc_bselres_e*=1) from the A-Stage (*mpc_bselall_e*=0).

EXERCISE 2:

- **Bubble insertion**:

  1. When instruction (i-1) is a lw with destination register Rx, and instruction (*i*) is an Arithmetic-Logic instruction with source register Rx, one cycle must be inserted between both instructions.

  2. For creating a bubble at Stage-E, ***mpc_run_ie***=0. This signal is assigned from many signals and logic (***(prod_cons_stall_req*** & ***ldst_m)*** → ***load_to_use_stall_e*** → ***prod_cons_stall_e*** → ***bubble_e*** → ***bstall_ie*** → ***stall_ie*** → ***mpc_run_ie***). Simplifying, it is 0 when (***prod_cons_stall_req*** & ***ldst_m***) is 1.
     a. ***prod_cons_stall_req***:It is set when there is a RAW dependency between the instruction at Stage-M and the instruction at Stage-E.
     b. ***ldst_m***:It is set when the instruction at Stage-M is a load or a store.

  3. Signal ***mpc_run_ie***=0 has many effects, such as the following:

     - The pipeline registers from E-Stage to M-Stage (*_ie_pipe_out_50_0_*) do not change:

```
mvp_cregister_wide #(51)
_ie_pipe_out_50_0_(ie_pipe_out[`M14K_IE_RANGE],gscanenable, mpc_run_ie,
gclk, ie_pipe_in);
```

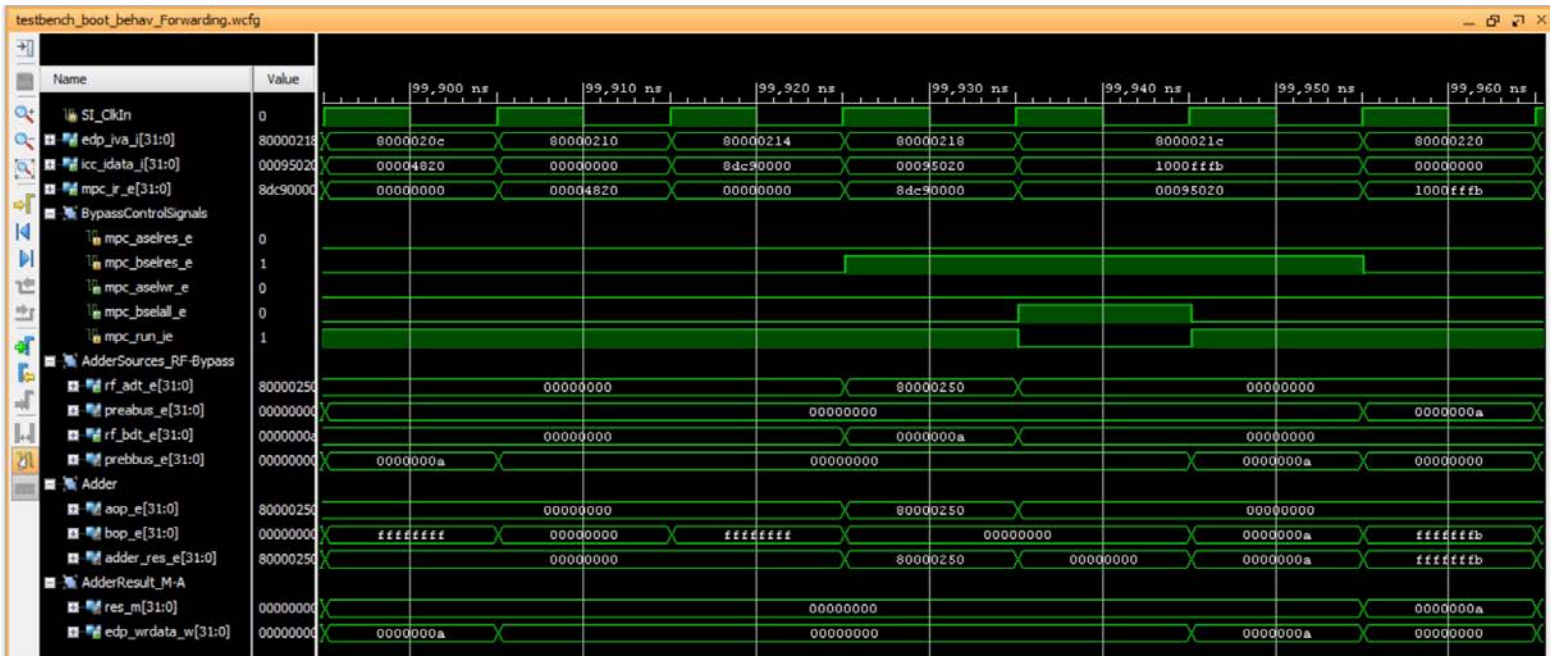
     - The IR (***mpc_ir_e*** ← ***int_ir_e***) is not updated:

```
mvp_cregister_wide #(32) _int_ir_e_31_0_(int_ir_e[31:0],gscanenable,
(mpc_run_ie | mpc_fixupi) & ~mpc_atomic_e
& ~mpc_lsdc1_e, gclk, icc_idata_i);
…
assign mpc_ir_e[31:0] = sel_hw_e ? hw_ir_e : int_ir_e;
```

     - The PC at Stage-I (***edp_iva_i***) is not updated:

```
mvp_cregister_wide #(32) _edp_iva_i_31_0_(edp_iva_i[31:0],gscanenable,
(!icc_umipsmode_i & mpc_run_ie) ||( icc_umipsmode_i & !(icc_imiss_i &
~(mpc_hw_ls_i | mpc_chain_hold)) & (mpc_run_ie | mpc_fixupi)), gclk,
edp_iva_p_exit);
```

SIMULATION:



In the fifth cycle a dependency is detected between the `lw` instruction at the M-Stage and the second operand of the `add` instruction at the E-Stage. Given that the result of the `lw` instruction is not yet available, the `add` instruction is stalled at the E-Stage.
In the sixth cycle, the second operand of the `add` instruction is provided through forwarding (*mpc_bselres_e*=1) from the A-Stage (*mpc_bselall_e*=0).