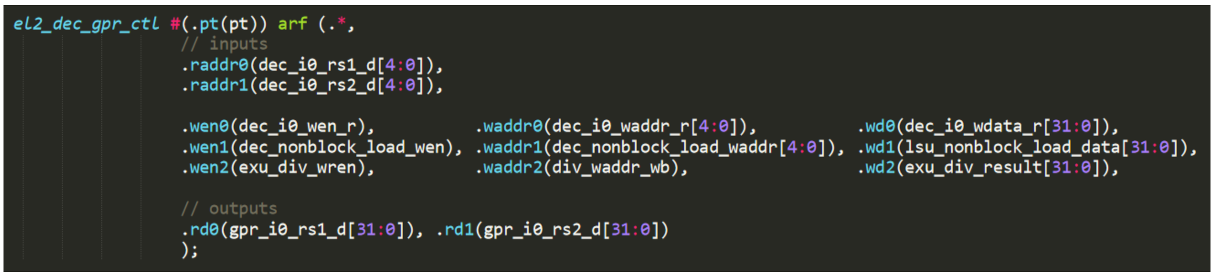
TASK: The Register File is implemented in module **el2_dec_gpr_ctl** and it is instantiated in module **el2_dec** (see **Error! Reference source not found.**). Analyse both the Verilog code and the simulation of the main signals of module **el2_dec_gpr_ctl** (available in file *[RVfpgaEL2NexysA7NoDDRPath]/src/VeeRwolf/VeeR_EL2CoreComplex/dec/el2_dec_gpr_ctl.sv*), in order to understand how it works.

Instantiation in module dec:

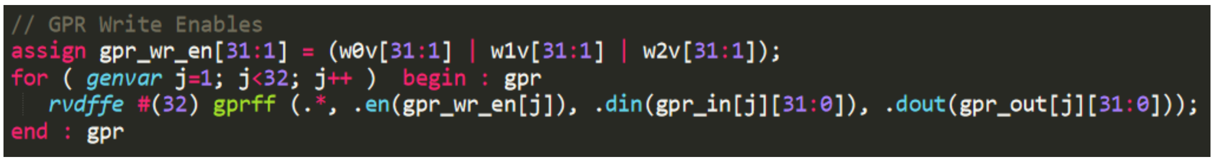


```
el2_dec_gpr_ctl #(.pt(pt)) arf (.*,
                    // inputs
                    .raddr0(dec_i0_rs1_d[4:0]),
                    .raddr1(dec_i0_rs2_d[4:0]),

                    .wen0(dec_i0_wen_r),            .waddr0(dec_i0_waddr_r[4:0]),           .wd0(dec_i0_wdata_r[31:0]),
                    .wen1(dec_nonblock_load_wen), .waddr1(dec_nonblock_load_waddr[4:0]), .wd1(lsu_nonblock_load_data[31:0]),
                    .wen2(exu_div_wren),            .waddr2(div_waddr_wb),                  .wd2(exu_div_result[31:0]),

                    // outputs
                    .rd0(gpr_i0_rs1_d[31:0]), .rd1(gpr_i0_rs2_d[31:0])
                    );
```

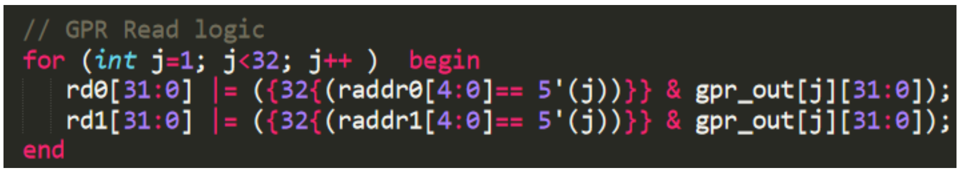
Implementation of the 32 registers in module dec_gpr_ctl:



```
// GPR Write Enables
assign gpr_wr_en[31:1] = (w0v[31:1] | w1v[31:1] | w2v[31:1]);
for ( genvar j=1; j<32; j++ )  begin : gpr
   rvdffe #(32) gprff (.*, .en(gpr_wr_en[j]), .din(gpr_in[j][31:0]), .dout(gpr_out[j][31:0]));
end : gpr
```

31 registers are implemented by instantiating 31 times module **rvdffe** (which you can find in file *[RVfpgaEL2NexysA7NoDDRPath]/src/VeeRwolf/VeeR_EL2CoreComplex/lib/beh_lib.sv*). Note that the width of each **rvdffe** register is selected using a parameter, which in our case is 32 bits → rvdffe #(32). Register 0 is not necessary as RISC-V architecture forces it to be always 0.

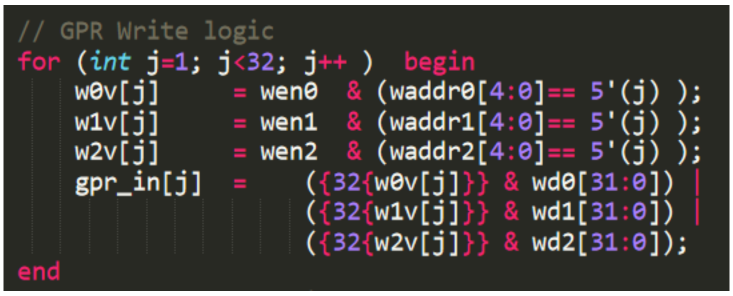
Register reading:

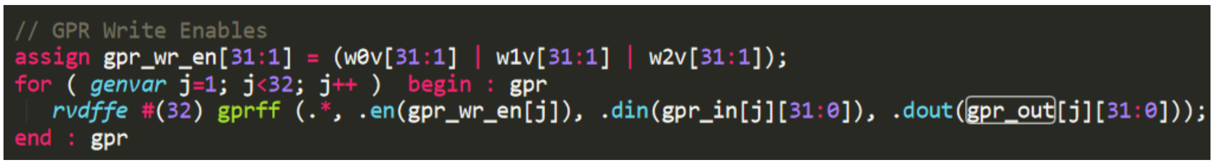


```
// GPR Read logic
for (int j=1; j<32; j++ )  begin
   rd0[31:0] |= ({32{(raddr0[4:0]== 5'(j))}} & gpr_out[j][31:0]);
   rd1[31:0] |= ({32{(raddr1[4:0]== 5'(j))}} & gpr_out[j][31:0]);
end
```

2 read ports are implemented. Each one is assigned in the `rd0/rd1` signals with the value of the register indicated by the `raddr0/raddr1` signals. Note that the initial value of *j* is 1, thus the reading of register 0 always returns the value 0.

Register writing:


```
// GPR Write logic
for (int j=1; j<32; j++ )  begin
    w0v[j]     = wen0  & (waddr0[4:0]== 5'(j) );
    w1v[j]     = wen1  & (waddr1[4:0]== 5'(j) );
    w2v[j]     = wen2  & (waddr2[4:0]== 5'(j) );
    gpr_in[j]  =     ({32{w0v[j]}} & wd0[31:0]) |
                     ({32{w1v[j]}} & wd1[31:0]) |
                     ({32{w2v[j]}} & wd2[31:0]);
end
```


```
// GPR Write Enables
assign gpr_wr_en[31:1] = (w0v[31:1] | w1v[31:1] | w2v[31:1]);
for ( genvar j=1; j<32; j++ )  begin : gpr
   rvdffe #(32) gprff (.*, .en(gpr_wr_en[j]), .din(gpr_in[j][31:0]), .dout(gpr_out[j][31:0]));
end : gpr
```

3 write ports are implemented. Each register is written with the value provided in signals `wd0/wd1/wd2`, depending on the register address `waddr0/waddr1/waddr2`. The `wen0/wen1/wen2` signals enable/disable the writing. Note that the initial value of *j* is 1, thus there is no write of register 0.

TASK: Execute the program on the Nexys A7 board as explained in the GSG. You should obtain the results shown in **Error! Reference source not found.** for the four measured events. Explain and justify the results.

The program is made up by a loop with 6 instructions, which performs 1.000.000 iterations. Ideally it would take 6.000.000 cycles to execute; however, 1 cycle is lost per iteration due to the misprediction of the loop branch (remember that the Branch Predictor is disabled in this example).

TASK: Execute the program on the RVfpgaEL2-ViDBo simulator. You should obtain the same results as those when the program is executed on the board.

Same solution as previous task.

TASK: Measure other events in the Hardware Counters for the same program. For this purpose, you must change in file *Test.c* the configuration of the events to be measured. Note that the different events (shown in **Error! Reference source not found.**) can be configured using the macros defined in the PSP file *psp_performance_monitor_eh1.h*. For example, if you want to measure the number of I$ misses instead of the number of branch misses, you must substitute in file *Test.c* line:

```
  pspMachinePerfCounterSet(D_PSP_COUNTER3, D_BRANCHES_MISPREDICTED);
```
for line:
```
  pspMachinePerfCounterSet(D_PSP_COUNTER3, D_I_CACHE_MISSES);
```

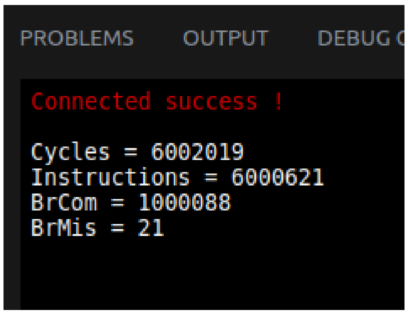
Solution not provided for this exercise.

TASK: Propose other programs in the `Test_Assembly` function and check if the different events provide the expected results. You can try other instructions such as loads, stores, multiplications, divisions… as well as hazards that provoke pipeline stalls.

Solution not provided for this exercise.

TASK: Test again the program provided at *[RVfpgaEL2NexysA7NoDDRPath]\Labs\Lab11\HwCounters_Example* and used in the previous section. In this case, enable the Gshare branch predictor and explain the results. You can execute it both on the physical board and on RVfpgaEL2-ViDBo.

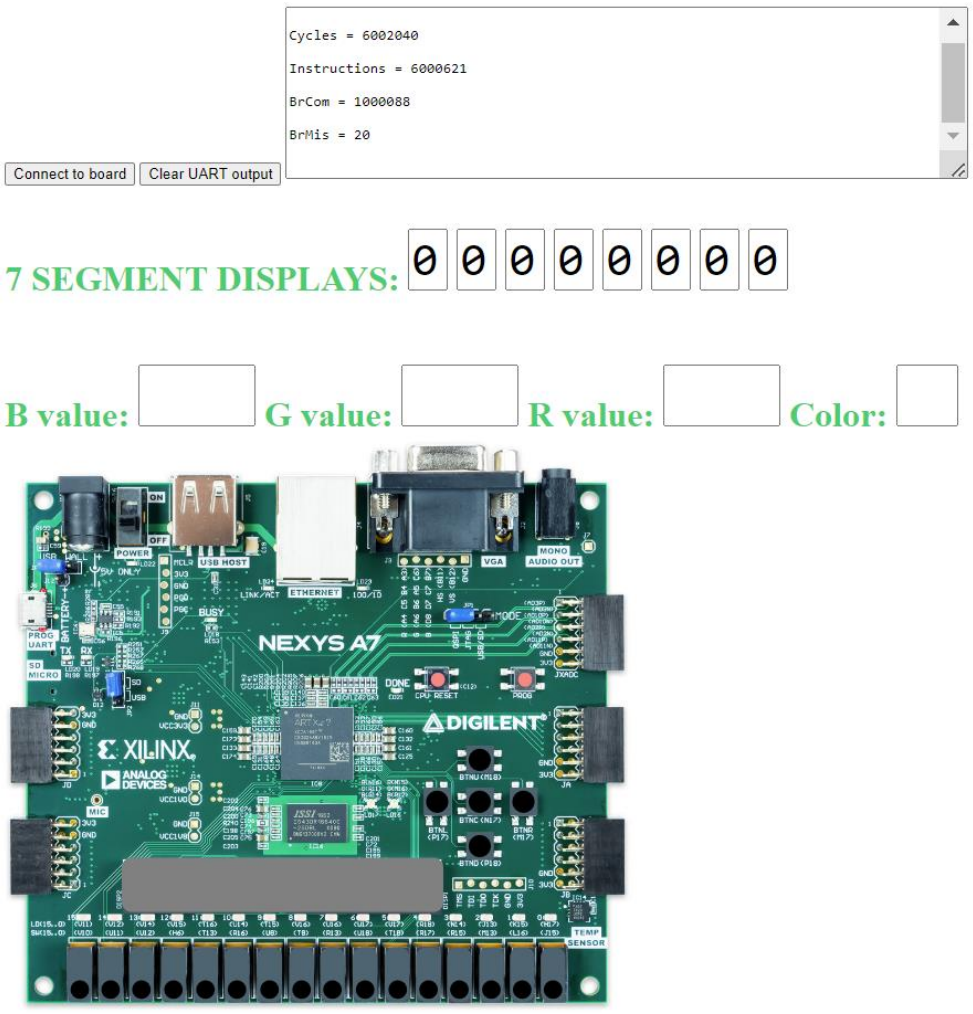
Execution on the board:



```
Connected success !

Cycles = 6002019
Instructions = 6000621
BrCom = 1000088
BrMis = 21
```

Execution on the RVfpgaEL2-ViDBo:



```
Cycles = 6002040

Instructions = 6000621

BrCom = 1000088

BrMis = 20
```

When the Gshare BP is enabled, IPC achieves its optimal value of 1.