HOMEWORK-7

1. Designing the Control Word

Design a control word for your processor by considering instruction structures of RV32I set as shown on page 18 of "ITU_VLSI_II_RF_Datapath_SingleCycle_Control.pdf" lecture slides. The length and the format of the control word will differ from the slides because you're using RISC-V ISA.

a. Determine what control signals you will need in order to implement RV32I. According to this, draw your control word structure as shown on page 18 of "ITU_VLSI_II_RF_Datapath_SingleCycle_Control.pdf" lecture slides.
b. Produce control word encoding table as shown on page 20 of "ITU_VLSI_II_RF_Datapath_SingleCycle_Control.pdf" lecture slides.
c. Give explanations and justifications regarding to your work. Briefly explain why you need each specific control signal.

2. Forming the Datapath

Make required connections between the modules you designed in previous homeworks; Register File (RF), Function Unit (FU) and multiplexers as shown on page 6 of "ITU_VLSI_II_RF_Datapath_SingleCycle_Control.pdf" lecture slides. Note that your RF will be sized 32x32 as required by RV32I set. Give the same names to the signals as much as possible in order to make traceability easier for us. Your top module should be named as "Datapath.v" and look similar to the grey box on page 28 of "ITU_VLSI_II_RF_Datapath_SingleCycle_Control.pdf" lecture slides. Be aware that the slides just introduce an example, you will have a different architecture for RISC-V in the end.

a. Check out the algorithm given at Figure 1. When the rows starting with "if" and "for" keywords are removed and C is initialized to 1, we get the following operations:
   C=1, C=2C, C=C-N, C=C+A, C=C-N.
   Assume that "A" and "N" are located in the **Data Memory,** and C will be formed from scratch in one of the registers. State that by which instructions you can perform these operations in given order. Produce needed control word and input signal list in order to implement these five operations in the given order.
b. Write a testbench in order to apply the list which you produced in Step 1.b to your DP. Instantiate the DP first, then instantiate the Data Memory that you've written in Homework 5 in your testbench with word size 32 and with an arbitrary depth (do not synthesize it) and connect it to your DP. You may initiate your memory with a text file using **$readmemb** function, as you did in Homework 5.
c. Perform behavioral simulation using the testbench you created in previous step. Test your circuit with 3 different pairs of {A,N}. Pick A and N in such a way that you can obtain a positive valued C, a negative valued C and C=0 in the end. Present and explain your simulation results.

3. Designing Instruction Decoder

Design Instruction Decoder (ID) block of your processor which translates the instructions to the control world as shown in lecture slides page 28 of "ITU_VLSI_II_RF_Datapath_SingleCycle_Control.pdf" slides.

a. Design ID block as shown in lecture slides page 28 of "ITU_VLSI_II_RF_Datapath_SingleCycle_Control.pdf" slides.

b. Perform **behavioral simulation** for your ID block. Apply the instructions you come up with at Step 1-b to your ID input and check if it is generating the signals you expected. Present and explain your simulation results.

4. Program Counter

Normally, the program counter (PC) must count up 4. But, it must also have branch and jump control inputs as shown in lecture slides page 28 of "ITU_VLSI_II_RF_Datapath_SingleCycle_Control.pdf" slides.

a. Design branch controller as shown in lecture slides page 28 of "ITU_VLSI_II_RF_Datapath_SingleCycle_Control.pdf" slides.
b. Design PC module as shown in lecture slides page 28 of "ITU_VLSI_II_RF_Datapath_SingleCycle_Control.pdf" slides.
c. Synthesize the top-level code of your PC by using OpenLane flow.
d. Perform **post-synthesis simulation** for the PC module. Aim to test all functionalities of the PC. Present and explain your simulation results.

5. Designing control unit (CU).

a. Integrate the PC and the Instruction Decoder in order to design the CU. As stated in Step 2, **Instruction Memory** will be connected over from a testbench, so include any required memory I/O ports in your CU as well. For **immediate generation**, refer to the table given at *RISC-V Instruction Set Manual, page 17*. Write a section within your CU that generates immediate values according to this table.
b. Synthesize the top-level code of your CU by using OpenLane flow.
c. Create a testbench and connect your CU and Instruction memory, similar to step 2-c. Return to the list of instructions you derived for the algorithm at Figure 1 at step 2-b. By adding the if and for constructs to your code from Step 2-a, implement the whole algorithm at Figure 1 and write your machine codes on a text file. Alternatively, you may prepare an arbitrary initialization file for your instruction memory that will include at least one instruction from all instruction format types. By using **$readmemb** function, load this file to Instruction memory, as done in Homework 5.
d. Perform **post-synthesis** simulation for your CU. Aim to test all functionalities of your CU. Present and explain your simulation results.

Note: If any changes are needed in your FU design from the previous homework, please correct them in this homework.

Left-to-right modular multiplication algorithm
Input: $A=(a_{k-1},a_{k-2},\cdots,a_1,a_0)_2$, $B=(b_{k-1},b_{k-2},\cdots,b_1,b_0)_2$, $N=(n_{k-1},n_{k-2},\cdots,n_1,n_0)_2$
Output: $C=AB \bmod N=(c_{k-1},c_{k-2},\cdots,c_1,c_0)_2$

```
Step1:  C=0
Step2:  for i=k-1:0
Step3:          C=2C
                if C≥N
Step4:                  C=C-N
Step5:          if bi=1
Step6:                  C=C+A
                        if C≥N
Step7:                          C=C-N
```

Figure 1