Question #1

Explain what the LedsSwitches_functions.c program does?

The program flushes the switches val into the leds of our fpga card.

Question #2

1. What is the value of the PC? What is it used for?

Pc = 0x58, it is used as an instruction pointer, to get the current instruction address.

2. What is the value of the SP register? What is it used for?

SP = 0x04000000, it is the stack pointer.

3. What is the value of the RA register? What is it used for?

RA = 0x50, it is the return address.

Question #3

Execute the Program by clicking **Step into F11** 5 more times

At each step, write down the registers and memory that change, and explain.

At first – when calling IO Setup, we jump tp the new func label, PC is changed due to the jump (using jal, we also save the return address in ra).

Before executing the func, we allocate and move parameters, this is done by allocating space in stack pointer and putting the needed value in a0 and storing it in the new stack range.

In the function body we load this param and execute the function

At last, we return stack pointer to its initial value and return to the caller address via ret command.

Question #4

Open the **LED-Switch_C-Lang.c** code and explain what it does.

The code Uses interrupts to read the value of Switch[0] and invert the state of the right-most LED.
We will generate an interrupt on every rising edge (0 → 1 turns on) of the switch.

Question #5

1. Write down how many times you slid the switch:

11.

2. Copy the results from Terminal Emulator (the calc and DELAY loops are not important, look at SW0):

calc = 19

SW0 0->1 = 11

DELAY loops 524287 done = 19

Question #6

1. Write down how many times you slid the switch:

6.

2. Copy results from Terminal Emulator :

calc = 19

SW0 0->1 = 11

DELAY loops 524287 done = 19

calc = 19

SW0 0->1 = 2

DELAY loops 524287 done = 19

3. Explain the results in Question #5 and Question #6

   what is wrong and why?

In polling, it looks like the program didn't "catch" all interrupts, this is due to the fact polling is not "invoked" like interrupts, thus making it miss some switch changes.

Question #7

1. Copy results from Terminal Emulator :

   calc = 19

   SW0 0->1 = 8

   DELAY loops 524287 done = 19

2. Explain the results.

In this case, we are in an “Active interrupts”, so whenever the switch is flipped the processor leaves everything and takes care of the interrupter.

3. Compare the results of questions 6 and 7, and list the advantages and disadvantages of a small and large delay in both scenarios. What is the effect of the delay on the sampling operation of the switch in each scenario?

In the interrupt’s method, we take care of each interrupt and don't miss important ones, but we have a lot of overhead due to the fact we stop other calcs the processor doing.
in polling we miss some interrupts because the processor doing other calcs and cannot catch all the triggers quickly.

Question #8

Follow all the steps in the simulation and provide a detailed explanation about the flow of Instruction:

```
add t5, t5, 1          # 001F0F13
```

through the pipeline.

Explain all pipeline steps.

In the first stage, we fetch the command, its opcode can be seen in ifu_io_instrc, which means it will be done in pipe 0.

then we can see the command moving to decode stage (dec_i0_inst_d), from there moving in the next cycle to exc1 and staying there for 4 cycles (as it is an this possessor arch).

From there to wb stage (committing the changes done to the register).

Question #9

Follow all the execution steps, examine, and explain:

1. At which stage in the pipeline is the branch instruction identified?

Decode (2).

2. Where is the decision to flush the pipeline made?

Ex1 (3).

3. At which stage in the pipeline is the flush signal activated?

Ex2 (4).

4. What happens in the pipeline as a result of the flush signal?

The pipe is flushed, thus all its controls are signals zeroed and the pc is changed again.

5. How many times is the flush activated? Explain why.

11 (10 as result of beq t3, t3, LOOP # FFCE00E3 & 1 as a result of beq t3, t4, OUT # 01de0e63).

6. Do the instructions **sub t3, t1, t1** and **sub t4, t1, t1** enter the pipeline, and how many times? Are they executed?

Yes, they enter 10 times, executed none.

7. How many cycles are lost?

4 * 10 = 40 cycle (as a result of 10 as result of beq t3, t3, LOOP # FFCE00E3) + 4 * 1 =4 (as a result of beq t3, t4, OUT # 01de0e63).

8. Write down the time when the value of the T6 register changes to 8.

11.65 ns

Question #10

Follow all the steps of execution, examine the explain:

1. How many times is the flash activated? Explain why.

11 times, as a result of that fact the two branches conflict in run time (one is taken always when the iteration runs and one is not taken) thus making the Gshared "noisy" making it not very effective.

2. Do the instructions **sub t3, t1, t1** and **sub t4, t1, t1** enter the pipeline and how many times? Are they executed?

No, since Gshared will encounter beq t3, t3, LOOP # FFCE00E3 when he is on "taken" mode, thus will not put these not instruction into the pipe.

3. How many cycles do we lose?

10 * 4 = 40 cycles

4. Write down when the R6 register changes its value to 8 and compare it to the previous question.

10.67ns.

Question #11

Write down the results received in the terminal:

Cycles = 871

Branch Miss = 32

Question #12

Write down the results received in the terminal:

Cycles = 822

Branch Miss = 15

Compare to the previous result and explain.

The previous result is without branch prediction, resulted in many branch miss prediction as a result of branch prediction disable (Branch Miss = 32 – every branch predicted not taken always), while the current result with branch prediction enable decreased the number of the branch miss prediction (Branch Miss = 15), as well the cycles number is lower in case branch prediction enable from the same reason (the flush's decreased).

Question #13

Open the Test.c program and explain what it does:

the test-sets up counters for the benchmark, it is also modifying the core features (according to what we want – in this example Disable Branch Predictor) and then it waits for they users to flip any switch to start the test.

Question #14

Fill out the table:

we will fill using the output

	DDR2 Memory	DCCM Memory
Cycles	**1726266**	**473591**
Instructions	**404997**	**403971**
Data Bus Transactions	**109236**	**4850**
Inst Bus Transactions	**616**	**632**

Question #15

Complete the table from the previous question for the new run.
Compare the results and explain.

Exercise

This is our code, we added a data hazard.

```
.globl main

.text

main:

# Disable Branch Predictor:

# csrrs command reads the current value of the CSR at address 0x7F9,

# stores it in register t1, and sets the bits in that CSR based on the contents of register t2.

li t2, 0x008      # Disable Branch Predictor

csrrs t1, 0x7F9, t2

li t3, 0x10  # number of loop iterations

li t4, 0x1   # loop counter

LOOP:

  addi t4, t4, 1

  beq t3, t4, OUT    # 01de0e63

  sub t3,t1,t1      # 40630e33

  sub t4,t1,t1      # 40630eb3

  beq t3, t3, LOOP    # FFCE00E3

OUT:

  addi t6, t6, 8      # 008F8F93
```