RISC-V Single-Cycle and Pipelined Implementation: Control Path, Data Path and Hazards

In this exercise you will learn about the data and control path of the RISC-V processor and how to detect and resolve hazards.

Task 1. Getting to know Ripes

Open Ripes with the command `ripes` from the terminal. Execute the following program with the Single-Cycle and the Five Stage RISC-V processor.

```
1 li t0, 4
2 loop:
3     addi t0, t0, -1
4 bne zero, t0, loop
5 addi t1, t0, 33
6 sub t0, t0, t1
```

Choose right-click→Port→Show all values.

Which difference can you see between the two processors? Hint: What happens to the instructions after `bne zero, t0, loop` ?

Task 2. Implementing Dot Product

Implement the function `dot` stated below as RISC-V assembly program for the following processor variants from Ripes.

1. Single-Cycle RISC-V
2. Five Stage RISC-V
3. Five Stage RISC-V w/o forwarding or hazard detection
4. Five Stage RISC-V w/o hazard detection

```
1  int A[5] = {1, 2, 3, 4, 5};
2  int B[5] = {5, 4, 3, 2, 1};
3
4  int dot(int* A, int* B, int n){
5      int result = 0;
6      for(int i=0; i < n; i++){
7          result += A[i] * B[i];
8      }
9      return result;
10 }
11
12 int main(){
13     dot(A, B, 5);
14     return 0;
15 }
```

Follow these steps for the implementation of `dot` for each of the 4 processor variants:

1. Reserve memory in the `.data` part of your program for arrays A and B, and initialize them with the correct values.
2. Implement the `dot` function and verify the result.
3. If you detect any hazard, try to solve it by adding *no-operation* instruction (assembler instruction: `nop` → `addi zero, zero, 0`), then take a screenshot and mark the datapaths that are used.
4. Always use the minimum number of `nop`s.

Task 3. Control Hazard

Check your code for control hazards. In the Ripes implementation these are resolved by the processor flushing the pipeline. You will see the word flush next to an instruction:
nop (flush)

Find the sections in your code where a control hazard occurs