1. Analytical study in MIPSfpga

When the baseline program is executed in MIPSfpga, there are four stalls per iteration (2 for the RAW hazard between the `lw` and the `add` instructions, and 2 for the `nop`s after the `beq` and the `bne`).

Detailed analysis:

- 100 iterations are performed.
- The first instruction is fetched in cycle 1.
- The first `lw` is fetched in cycle 8 in the first iteration, and it is fetched in cycle 22 in the second iteration (note that the `beq` within the loop body is taken in the first iteration, as $t3 is zero). Thus iteration 1 needs 14 cycles.
- The second `lw` is fetched in cycle 22 in the second iteration, and it is fetched in cycle 37 in the third iteration (note that the `beq` within the loop body is non-taken in the 2-100 iterations, as $t3 is different than zero). Thus iterations 2 to 100 need 15 cycles.

➔ Cycles = 7 + 14 + 15*99 + 4 = 1510

➔ Instructions = 7 + 10 + 11*99 = 1106

➔ CPI ≈ 1.365

If we reorder the program, we obtain a CPI approximately equal to 1.

2. Empirical study in MIPSfpga

a. Original program (no optimizations)

The results provided by the performance counters are the following:

- **Number of cycles = 1533**
- **Number of instructions =** 1314 – 200 (2 nop per iteration) **= 1114**
  We need to correct the number of instructions provided by the performance counters, as nop instructions should not be included.
- **CPI =** 1533 / 1114 **≈ 1.37**

b. Reordered program

```
".set noreorder;"
"   addi    $t1,$0,100;"
"   lui     $t6, 0x8000;"
"   addiu   $t6, $t6, test_arrayA;"
"   lui     $t7, 0x8000;"
"   addiu   $t7, $t7, test_arrayB;"
```

```
"   lui     $t8, 0x8000;"
"   addiu   $t8, $t8, test_arrayResult;"
"LOOP:  LW      $t2,0($t6);"
"       ADDI    $t6,$t6,4;"
"      ADD  $t2,$t2,$t2;"
"       LW      $t3,0($t7);"
"       ADDI    $t7,$t7,4;"
"      ADD  $t3,$t2,$t3;"
"      BEQ       $t3,$0, NoAct;"
"       ADDI    $t1,$t1,-1;"
"       SW      $t3,0($t8);"
"NoAct: BNE     $t1,$0,LOOP;"
"       ADDI    $t8,$t8,4;"
".set reorder;"
```

The results provided by the performance counters are the following:

- **Number of cycles = 1129**
- **Number of instructions = 1112**
- **CPI =** 1129 / 1112 **≈ 1**

By reordering some instructions we are able to avoid the two bubbles after the `lw` instructions and to fill the delay slot of the conditional branches, thus reducing the CPI very close to the optimal value (CPI≈1).