1. Analytical study in the pipelined processor from [2]

The program contains: # instructions = 3 + 5*1000 + 2 = **5005 instructions**

As for the amount of cycles required: # cycles: It takes 3 + 7*1000 + 3 = **7006 clock cycles** to issue all the instructions, as there is 1-cycle stall due to the slt-beq data hazard, plus a 1-cycle stall due to the `j` instruction.

Finally, we can compute the CPI as: CPI = 7006 clock cycles / 5005 instructions = **1.4**

2. Analytical study in MIPSfpga

a. Original program (no optimizations)

The program contains 3 + 5*1000 + 2 = **5005 instructions**

As for the amount of cycles needed for the execution of the program in microAptiv:

- The delay slot is filled by the assembler with a *nop* instruction by default, thus each iteration has 2 bubbles (one after each *branch/jump* instruction), requiring 5+2 cycles (except the first one, that suffers some I$ misses).
- There are 3 instructions before the loop, which can also experiment an I$ miss, and 2 more instructions when leaving the loop, as *slt* and *beq* are executed once again after the 1000th iteration.

Taking all this into account: (3 + 7*1000 + 2 + I$_Miss_Delay) **= (7005 + I$_Miss_Delay) cycles**

The **CPI** would be approximately the same as the one obtained in the previous section, i.e. **1.4**.

b. Original program (-O3 option)

The compiler generates the following program, where it has been able to fill the delay-slot of the `j` instruction but not the delay slot of the `beqz` instruction:

```
80000314:   00008020    add   s0,zero,zero
80000318:   00008820    add   s1,zero,zero
8000031c:   200803e8    addi  t0,zero,1000
80000320 <loop>:
80000320:   0208482a    slt   t1,s0,t0
80000324:   11200004    beqz  t1,80000338 <done>
80000328:   00000000    nop
8000032c:   02308820    add   s1,s1,s0
80000330:   080000c8    j     80000320 <loop>
80000334:   22100001    addi  s0,s0,1
```

c. Reordered program

In our program we can successfully fill both delay slots by doing some rearrangements. The new program is the following:

```
.set noreorder
add $s0, $0, $0 # i = 0
add $s1, $0, $0 # sum = 0
addi $t0, $0, 10 # $t0 = 1000
loop:
      slt $t1, $s0, $t0 # if (i < 1000), $t1 = 1, else $t1 = 0
      beq $t1, $0, done # if $t1 == 0 (i >= 1000), branch to done
      add $s1, $s1, $s0 # sum = sum + i
      j loop
      addi $s0, $s0, 1 # increment i
done:
sub $s1, $s1, $s0 # sum = sum – i
.set reorder
```

Note that we have performed several changes to the original program:

- We are including directives *.set noreorder* (before) and *.set reorder* (after).
- We have exchanged "*j loop*" and "*addi $s0, $s0, 1*" instructions, so that the second one is executed in the delay slot of the *jump* instruction.
- Instruction "*add $s1, $s1, $s0*" is now executed in the delay slot of the *beq* instruction. This is not a problem, except after the last iteration, when the *beq* is taken and another *add* instruction is executed although it shouldn't. For solving that problem, we have included a new *sub* instruction right after the loop, which undoes the incorrect extra addition.

The program contains 3 + 5*1000 + 3 + 1 = **5007 instructions**

As for the amount of cycles, only (3 + 5*1000 + 3 + 1 + I$_Miss_Delay) **= (5007 + I$_Miss_Delay) cycles** would be required.

In this case, The **CPI** would be reduced to approximately **1**.

3. Empirical study in MIPSfpga

a. Original program (no optimizations)

The results provided by the performance counters are the following:

- **Number of cycles = 7025**
- **Number of instructions =** 7012 – 2*1000 (2 nops per iteration) – 1 (1 final nop) **= 5011**

We need to correct the number of instructions provided by the performance counters, as nop instructions should not be included.

- **CPI =** 7025 / 5011 ≈ **1.4**
  Note that this result is not perfectly accurate, as there are some extra instructions included due to the Performance Counter accounting process. You can look for "<main>:" in file *program.dis* to inspect the assembly code generated by the compiler.

b. Original program (-O3 option)

The results provided by the performance counters are the following:

- **Number of cycles = 6021**
- **Number of instructions =** 6008 – 1*1000 (1 nops per iteration) – 1 (1 final nop) **= 5007**
  We need to correct the number of instructions provided by the performance counters, as nop instructions should not be included.
- **CPI =** 6021 / 5007 ≈ **1.2**
  Note that this result is not perfectly accurate, as there are some extra instructions included due to the Performance Counter accounting process. You can look for "<main>:" in file *program.dis* to inspect the assembly code generated by the compiler.

c. Reordered program

The results provided by the performance counters are the following:

- **Number of cycles = 5026**
- **Number of instructions = 5014**
- **CPI =** 5026 / 5014 ≈ **1**
  Note that, like above, this result is not perfectly accurate, as there are some extra instructions included due to the Performance Counter accounting process.

Thus, in this example, using the delay slot allows us to **reduce the experimental CPI from around 1.4 (program with no optimization) or around 1.2 (program using –O3 optimization) to the optimal CPI of 1**.