The radical of a positive integer *n*, denoted rad(*n*), is defined as the product of the distinct prime factors of *n*. For example: rad(48) = rad(24 \* 3) = 2 \* 3 = 6.

The abc conjecture states that if *a*, *b*, and *c* are three positive integers such that:

Duration: 90 minutes.

It is possible to consult:

- any paper material

- slides downloaded from the course page on the teaching portal

- code of the laboratories, if uploaded to the teaching portal in the “elaborati” section.

Students caught communicating with each other will be immediately removed from laboratory.

* *a* + *b* = *c*
* they are coprime, i.e., the only positive integer that divides all of them is 1.

then “usually” *c* < rad(*a* \* *b* \* *c*).

Formally, for every real value of ε, there exists a constant *K* such that *c* < *K* \* rad(*a* \* *b* \* *c*)1 + ε.

The abc conjecture is regarded as the most important unsolved problem in the analysis of polynomial equation with integer solutions.

It is required to write a program to check the validity of the abc conjecture as follows:

1. create a new project with Keil inside the **template** directory
2. replace the contents of the startup\_LPC17xx.s file with the one in the **template** directory
3. create the group **main** in the Keil project and add the sample.c file inside
4. create other groups according to the subdirectories in the **template** directory that you need to import (not all of them may be needed for this exam).
5. write **debugged** and **working** assembly subroutines and C instructions in order to meet the following 3 specifications.

*Note 1*: You should not change the code calling the subroutines in the startup\_LPC17xx.s file. It is only required to implement the assembly subroutines.

*Note 2*: Specifications must be completed in order. You can only move to Specification 2 after verifying that the solution to Specification 1 is working correctly. Same for Specification 3.

*Note 3*: Assembly subroutines must comply with the ARM Architecture Procedure Call Standard (AAPCS) standard (about parameter passing, returned value, callee-saved registers).

**Specification** **1** (8 points). Write a radical subroutine that computes the radical of a positive integer. The subroutine receives the integer in input and returns its radical in output.

*Suggestion*: in order to compute the radical of integer *x*, use a loop with an index initialized to 2.

If the index is not an exact divisor of *x*, increment the index for the next iteration of the loop.

If the index is an exact divisor of *x*, divide *x* by the index. Then, in the next iteration of the loop try dividing *x* by the same index again (i.e., do not increment the index for the next iteration). In order to find the radical, multiply each exact divisor of *x* only once (i.e., only the first time the exact divisor is found). The loop ends when *x* becomes 1.

**Specification 2** (7 points). Write a coprime subroutine that checks if two positive integers *u* and *v* are coprime. The subroutine receives the two integers in input and returns 1 if they are coprime, 0 otherwise. You should note that:

* if *u* and *v* are both even, they are not coprime
* if *u* is even and *v* is odd, they are coprime only if *u*/2 and *v* are coprime
* if *u* and *v* are both odd and *u* < *v*, then they are coprime if *u* and (*v* – *u*)/2 are coprime.

Therefore, you can use the following pseudocode:

if (u is even AND v is even)

return 0;

while (u is even)

u = u / 2;

do {

while (v is even)

v = v / 2;

if (u > v)

swap u and v;

v = v – u;

} while (v != 0);

if (u == 1)

return 1;

else

return 0;

Important: you can not use any operation of division for implementing the pseudocode. In particular, you can check if a number is even by testing if the least significant bit is 0. You can divide a number by 2 with a right shift.

**Specification** **3 (**4 points). Declare 3 variables *a*, *b*, *c* in the main() function; initialize *a* to 27 and *b* to 1. Write a loop in the main() function to count how many times *c* > rad(*a* \* *b* \* *c*) among the first 100 valid combinations of *a*, *b*, *c* values. In details:

1. check if the current combination of values (*a* = 27, *b* as incremented at every iteration of the loop, *c* = *a* + *b*) are coprime, by calling the coprime subroutine three times. The parameters are *a* and *b* at the first call, *a* and *c* at the second call, *b* and *c* at the third call.
2. If the current values of *a*, *b* and *c* are coprime:
   1. increment the counter of admissible solutions
   2. if *c* > rad(*a* \* *b* \* *c*), increment the counter of exceptions to the abc conjecture. Call the radical subroutine in order to compute rad(*a* \* *b* \* *c*).
3. Repeat the loop by incrementing *b*. The loop ends when the counter of admissible solutions reaches 100.
4. At the end of the loop, switch on the led corresponding to the number of exceptions:
   1. 0 exceptions -> led 4 on
   2. 1 exception -> led 5 on
   3. 2 exceptions -> led 6 on
   4. 3 exceptions -> led 7 on
   5. 4 exceptions -> led 8 on
   6. 5 exceptions -> led 9 on
   7. 6 exceptions -> led 10 on
   8. 7 exceptions -> led 11 on

You can assume that no more than 7 exceptions occur.