Computer Architectures
Exam of 28.01.2020 - Part II

B

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.

Given two unsigned (strictly positive) integers x and y, let z be their greatest common divisor. The Bézout's identity states that there are infinite pairs of signed integers C and D such that $Cx + Dy = z$.

It is required to write a program to compute a pair of coefficients in the Bézout's identity 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 `binaryGCD` subroutine that computes the greatest common divisor of two unsigned integers x and y. The subroutine receives x and y in input and returns z = gcd(x, y). You should note that:

- if x and y are both even, then gcd(x, y) = 2 * gcd(x / 2, y / 2)
- if x is even and y is odd, then gcd(x, y) = gcd(x / 2, y)
- if x is odd and y is even, then gcd(x, y) = gcd(x, y / 2)
- if x and y are both odd, then gcd(x, y) = gcd($x - y$, y) if $x > y$; gcd(x, y - x) if $y > x$.

Therefore, you can use the following pseudocode:

```
g = 0;
while (x is even AND y is even) {
     x = x / 2;
     y = y / 2;
     g = g + 1:
}
do {
     while (x is even)
          x = x / 2;
     while (y is even)
          y = y / 2;
     if (x > y)
          x = x - y;
     else
```

```
            y = y - x;
    } while (x != 0);
return v * 2^g;
```

Important: you can not use any operation of multiplication or 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. You can multiple a number by 2 with a left shift.

Specification 2 (7 points). Write a `binaryExtendedGCD` subroutine that receives in input two unsigned integers *x* and *y* and returns in output the coefficient *C* of the Bézout's identity. In order to compute *C*, it is suggested to copy and paste the subroutine developed at the previous point, and then implement the instructions marked in bold in the following pseudocode:

```
g = 0;
while (x is even AND y is even) {
      x = x / 2;
      y = y / 2;
      g = g + 1:
      }
do {
      u = x;
      v = y;
      A = 1;     /* Note: A, B, C, D are signed integers */
      B = 0;
      C = 0;
      D = 1;
      while (x is even) {
            x = x / 2;
            if (A is even AND B is even) {
                  A = A / 2;
                  B = B / 2;
            }
            else {
                  A = (A + v) / 2;
                  B = (B - u) / 2;
            }
      }

      while (y is even) {
            y = y / 2;
            if (C is even AND D is even) {
                  C = C / 2;
                  D = D / 2;
            }
            else {
                  C = (C + v) / 2;
                  D = (D - u) / 2;
            }
      }

      if (x > y) {
            x = x - y;
```

```
            A = A - C;
            B = B - D;
        }
        else {
            y = y - x;
            C = C - A;
            D = D - B;
        }
    } while (x != 0);
return C;
```

Specification 3 (4 points). Declare 2 variables *x* and *y* in the main() function; initialize *x* to 27 and *y* to 41. Compute *z* = gcd(*x*, *y*) and *C* by calling the two previously developed subroutines. Based on the *x*, *y*, *z*, and *C* values, compute *D*. Switch on and off leds in order to show the binary representation of the absolute value of *C* (if the absolute value of *C* is higher than 255, the value 255 is shown, i.e., all leds are switched on). In the binary representation, led 11 corresponds to the least significant bit of *C*.
After 2 seconds, show the binary representation of the absolute value of *D* (if the absolute value of *D* is higher than 255, the value 255 is shown). In the binary representation, led 11 corresponds to the least significant bit of *D*.