COM1006/COM1090 Devices and Networks (Autumn)
Tutorial Sheet #9: Assembly Language Programming

1. Download the following file from
   https://staffwww.dcs.shef.ac.uk/~dirk/campus_only/misc/factorials.X68
   and open it in EASy68K.

   a) Find out what the program does. Execute it step-by-step to see what each instruction does. Pay attention to how the status flags are being set after the `CMP` command. Don't miss the output in the console.

   b) Now modify the program (close the debugger to go back to the editor) so that instead of its current functionality it outputs all square numbers n^2 for n=1, 2, 3, ..., 10. Execute your program to test it. Note that square numbers must be put in D1 for the output to work.

2. Let's practise indirect addressing and how to deal with tables and strings. Download the stub file from
   https://staffwww.dcs.shef.ac.uk/~dirk/campus_only/misc/upper-case-stub.X68
   and open it in EASy68K. Your task is to convert a string "helloworld" at label `Text` to upper case.

   a) Load the effective address `Text` in A1 before the label `LOOP`. (We will use A1 to iterate through all characters in `Text`.)

   b) Add code between the label `LOOP` and the `JMP` command which transforms the current character at (A1) ([[A1]] in RTL) to upper case. Because of the ASCII table's structure (cf. Lecture 1, Slide 9), it suffices to clear the 6th bit, i.e. `AND`ing the character at (A1) with #%11011111. Make sure you use the correct size suffix for this.

   c) Extend your program so that after the code for b) the pointer A1 is increased by #1 ([A1]←[A1]+1 in RTL). This means that A1 now points to the next character in memory. This addition should be 32 bit.

   d) Implement a conditional branch to escape from the loop at the end of the string. The program should branch to the label `END` if A1 has reached the null-terminated end of the string (i.e. the byte at (A1) is zero). Refer to EASy68K's Help to find the right instruction.

   e) Test your program to see whether it works as intended, or use the debugger and step-by-step execution to find and correct any errors. (Note that the AND-trick in b) only works for alphabetic letters; in particular, blanks (0100000_2) are turned into zeros, so the string is cut off early!)

3. Compute the greatest common divisor (gcd) of two numbers P and Q using Euclid's algorithm. In Java-/Pseudocode it looks like this:

```
while(Q != 0) {         // while Q is not zero
      P = P % Q;        // P is set to the remainder of P/Q
      int temp = P;
      P = Q;            // exchange the values of P and Q
      Q = temp;
}
return P;               // at the end P contains the gcd.
```

Implement this algorithm in Motorola 68K assembly language, extending the stub source file from
https://staffwww.dcs.shef.ac.uk/~dirk/campus_only/misc/gcd-stub.X68
which provides some textual output.

As annotated in this file, use D2 for storing P and D3 to store Q. Recall that `DIVU` stores the remainder in the upper 16 bits and the result in the lower 16 bits of the target register.
Your program should branch to `OutputGCD` once Q=0.

Use another register as `temp` variable for exchanging P and Q, or look up the data movement command `EXG` in the Instruction Summary.

4. **[Optional encore]** Revise your solution for Exercise 2. so that it works for blanks and other characters like `!"§$%(){}` etc. as well. Add conditional branches so that the `AND` instruction is skipped if [[A1]] < 1100001_2 (ASCII code for 'a') or [[A1]] > 1111010_2 (ASCII code for 'z'). Test your program with an input containing blanks and various non-letter characters.