Lab 7
Microprocessor Architectures [ELEC-H-473]
SIMD 1: Image threshold

v1.0.1

Description of the lab

Objetives:

- Learn how to use Code::Blocks.
- Write elementary C programs and how to use SIMD with inline assembly.
- Learn how to use the debugging environment.
- For the proposed problem, implement C and SIMD versions of the program.
- Measure execution times to compare different implementations and understand the advantages of SIMD.

Preliminary

- Start by writing a “Hello World” program.
- Modify your code by adding some arithmetical operations.
- Launch the program in debug mode and execute it step-by-step.
- Learn how to observe register file change.

Turning a grey scale image black and white

Digital images can be represented in the form of a two-dimensional matrix where indexes i,j represent the spatial coordinates of the image. For a colour image there are three matrices, one per colour component R, G and B (Red, Green, Blue). For a grey level image, only one matrix is necessary (R, G, B components have the same value). Depending on image quantification, we can have more or less colours. Most of the time grey level images are coded using 8 bits of information per pixel. This means 256 colours with:

- Black is $(0)_{10} = (00000000)_2$
- White is $(255)_{10} = (11111111)_2$

In image processing applications, it is often useful to work with binary images that use only black and white colours. For such images, the pixel value is given only with a binary choice: black (0) or white (1). For images coded with 8 bits, this means 0 or 255 (0x00 or 0xFF in hexadecimal). In order to convert one grey level image into a black and white binary image we need to fix a threshold value. If the current pixel value is smaller than the threshold value, we will set the new pixel value to 0; otherwise it will be set to 255. Depending on the input image and the value of the threshold we will obtain different resulting images. For this exercise you can use the value of the threshold as you like, we are only interested in the way computation is being performed (and the execution time).

Here is the pseudo-code of what you will have to program:

```
for (all_images) {
        read_image_from_file
        start_time = get_time ()
        for (all_pixels_in_the_image) {
                if (curent_pixel_value < treshold) then new_pixel_value = 0
                else new_pixel_value = 255 }
        end_time = get_time ()
        out >> "Computing time " end_time - start_time
        write_image_result
}
```

To do so, you will need to include assembly inside your C code:

```
fonctionC()
{
        __asm__{
                /* ASM code */
                "sub $1, %%rdx\n"
        }
}
```

And here is a snippet of code measure time:

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

int main() {
        clock_t start, end;
        start = clock();

        /* Do something clever */

        end = clock();
        float time = (float)(end - start)/CLOCKS_PER_SEC;
        printf("Time spent: %f\n", time);
        return EXIT_SUCCESS;
}
```

Assignment

1. Implement the image threshold processing in plain C.
2. Implement the same algorithm in SIMD.
3. Benchmark both solutions and compare their performance. If the precision of your benchmarking is not sufficient (especially on Windows), run the filter several times (only the filter, not the file reading and writing).
   Comment on the expected acceleration factor, the actual results, the difference between debug and release compilation, etc.

Code requirements

– Send only your source files, *i.e.* .c/.h/.cpp/.hpp. Do *not* send the input and output files.
– If your code requires specific configuration flags for the compilation, please write them down as comments in your code.
– Make sure it's well documented.
– All the input and output files should be in the source directory.
– All the output files should have the same base name as the source files on the UV (ie. Escher, kid or lena_gray), appended with "out_C.raw" or "out_SIMD.raw".
– Do not use any weird library that may not compile on a Linux-based computer.