Arquitetura de Computadores 2023/24

TPC3

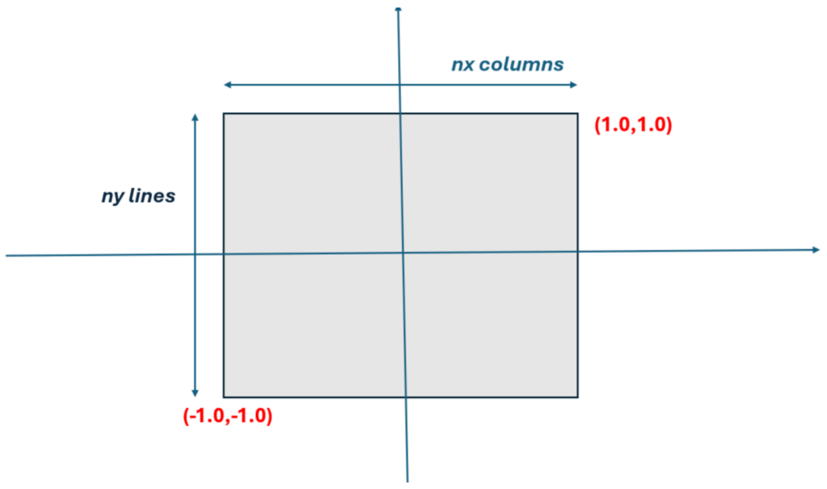
This homework consists of a programming exercise to be completed in **a group of no more than two students**. You can discuss general doubts with colleagues, but the solution and the writing of the code must be strictly carried out by the members of the group. All solutions will be automatically compared, and cases of plagiarism will be punished according to current regulations. **If you use tools such as CoPilot or ChatGPT, you must include a comment in the source code reporting such use**.

Deadline for submission: 14/5 (Tuesday) at 10:00.

Mandelbrot set

The Mandelbrot set has applications in various areas. In mathematics, it is used to study complex dynamic systems and fractal geometry. In physics, it is used to implement models to study natural phenomena. Images derived from the Mandelbrot set have applications in art. The set has also been used in data compression, cryptography, and other areas of computer science. What is remarkable about Mandelbrot's set is that very simple rules can produce infinitely complicated results.

In this work, we will utilize a combination of a C program and an x86-64 assembly program to generate a Mandelbrot set corresponding to a configuration described below:



The Mandelbrot set in question will be stored in a matrix with nx columns and ny rows. Being the coordinates of the lower-left vertex (xmin, ymin) and the upper-right vertex (xmax, ymax), we will have:

```
    delta_x = (xmax-xmin) / nx
    delta_y = (ymax-ymin) / ny
```

Each of the nx*ny points will have the following coordinates:

```
    x = xmin + delta_x * i
    y = ymin + delta_y * j
```

The value v corresponding to the point with coordinates (x, y) is given by the following algorithm in which x, y, zi, zr, nr, ni are real numbers (in double precision, in this work):

```
iterations = 0;
```

```
zi = 0;
zr = 0;
while ((zr*zr + zi*zi < 4.0) && (iterations < 255)) {
    nr = zr*zr - zi*zi + x;
    ni = 2*zr*zi + y;
    zi = ni;
    zr = nr;
    iterations++;
  }
  return iterations;
```

Program that calculates the Mandelbrot set

The program that calculates this Mandelbrot set is built from two source files:

- *mandel.c* This file is available on CLIP; it is complete and should not be modified. Its behavior can be described by the following pseudo code (see the code for more details):

  ```
  1. Obtain the image dimensions nx and ny and reserve a vector of bytes
     "buffer" with dimensions corresponding to a matrix with ny rows and
     nx columns
  2. for each buffer[x,y] element of the matrix
        a. calculate the real numbers x_value and y_value corresponding
           to the position column x, row y
        b. invoke the computePoint function with arguments x_value and
           y_value, which returns an integer value v between 0 and 255
        c. Invoke the updateImage function that sets buffer[x,y] =v
  3. create a file in PPM format with an image that allows visualizing
     the content of the matrix.
  ```

  The program is invoked with the command:

  ```
  ./mandel <num-columns> <num-lines>
  ```

- *mandel_ASM.* which must be created from scratch and defines the two functions computePoint and updateImage

The following Makefile should be used to generate the mandel executable

```
CC=gcc
CFLAGS=-g -z noexecstack
ASFLAGS=-gstabs
all: mandel
mandel: mandel.c mandel_ASM.o
	$(CC) $(CFLAGS) -o mandel mandel.c  mandel_ASM.o
mandel_ASM.o: mandel_ASM.s
	as $(ASFLAGS) -o mandel_ASM.o mandel_ASM.s
clean:
	rm -f *.o mandel *~
```

mandel_ASM.s

The mandel_ASM.s file should have the following structure:

```
.globl updateImage  #  void updateImage( buffer, x,     y,  val, base)
.globl computePoint #  unsigned char computePoint( double x, double y)
.section .note.GNU-stack,"",@progbits
.text
updateImage:

...
    retq

computePoint:

...
    retq
```

The functions to implement correspond to the signatures existing in the file mandel.c:

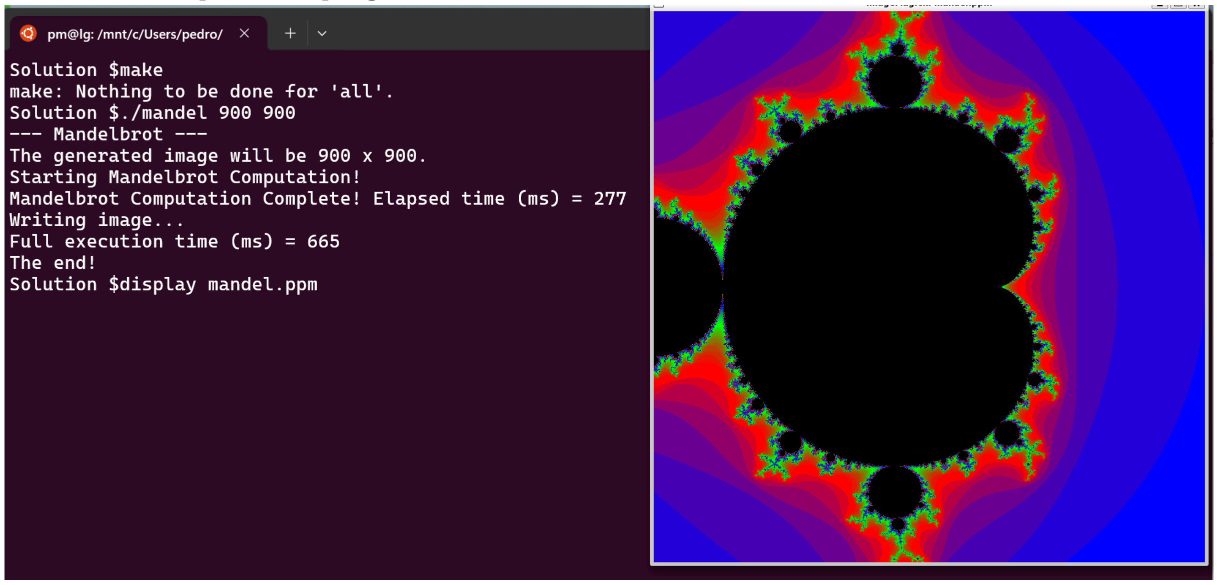
```
unsigned char computePoint( double cr, double ci );
void updateImage( unsigned char *buffer, unsigned int x,
       unsigned int y, unsigned char val, unsigned int base );
```

In the computePoint function you should use the machine instructions and SSE registers mentioned in the lectures and described in the document *CS:APP2e Web Aside ASM:SSE: SSE-Based Support for Floating Point* de Randal E. Bryant e David R. O'Hallaron available at *http://csapp.cs.cmu.edu/2e/waside/waside-sse.pdf*

The output_ppm function produces a file in PPM format.For information on this image format *https://en.wikipedia.org/wiki/Netpbm*. For the image to be generated, the color.map file must be available.
On Linux, to view the image, you can use the set of programs available if you install the ImageMagick package, which can be installed with the command: `sudo apt install imagemagick`
The file can be viewed from the graphical interface or from the bash with:

```
      display mandel.ppm
```

Here is an example of the program's execution:



```
Solution $make
make: Nothing to be done for 'all'.
Solution $./mandel 900 900
--- Mandelbrot ---
The generated image will be 900 x 900.
Starting Mandelbrot Computation!
Mandelbrot Computation Complete! Elapsed time (ms) = 277
Writing image...
Full execution time (ms) = 665
The end!
Solution $display mandel.ppm
```

Delivery method

The file with the x86-64 assembly functions should have the name mandel_ASM.s This file should be included in an email message to be sent to master Pedro Camponês's email address. (**p.campones@campus.fct.unl.pt**)

The subject of the message should be AC2024-TPC3 students XXXXX and YYYYY.
XXXXX is the student number of the 1st author and YYYYY is the number of the 2nd author. If the group has only one member, YYYYY must be 00000.