Programming Project: Drawing Circles

In the lecture and exercises you learned how to draw rectangles and the fractal of the Mandelbrot set. Here you will implement two algorithms to draw circles and compare them.

Important before you begin:

- Write the code as RISC-V assembler using the RV32-IMF instruction set without the use of a compiler
- You can use the RARS instruction set simulator provided either via the *CESP Docker container* or via the *Virtual CESP Server*.
- Use the register layout and labels given by the templates files (MANDATORY).
- Use the data types that are used in the pseudo code.
- All the code you implement is tested by the unit test in `test_circle_int.asm` and `test_circle_float.asm`. Do not change the content of these files.
- Criteria for grading are (among others):
  - Number of unit tests that are "PASSED"
  - Quality of comments in your code
  - The unit test files `test_circle_int.asm` and `test_circle_float.asm` MUST be runnable (no warnings or error messages) in RARS after you made changes to `circle_float.asm`, `circle_int.asm` and `draw_circle.asm`
  - Follow the register convention introduced in the lecture (register purpose, caller save, callee save, etc.)
  - Create a report (1 page) describing your implementation(e.g. realization, optimizations, etc.)
  - Your submission will be check automatically against other submissions and compiler results
  - The submission of a plagiarism represents an attempt to deceive.

Project Files

Filename	Description
Template files where you add YOUR code	
`circle_float.asm`	File to implement Task 1.1, 1.2, 1.4
`circle_int.asm`	File to implement Task 2.1
`draw_pixel.asm`	File to implement Task 1.3
Unit tests to verify results of the implementations	
`test_circle_float.asm`	Run to verify implementation of 1.1, 1.2, 1.4
`test_circle_int.asm`	Run to verify implementation of 2.1
Library files (only for `.include` directives)	
`display_base_address.asm`	Defines base addresses for display
`sinlut.asm`	Contains lookup table with sin values as IEEE float
`unittest_intfloat.asm`	Library with unit test function

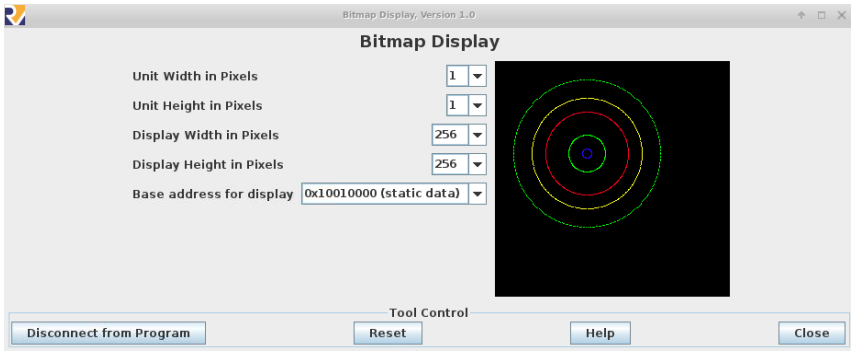

Figure 1: RARS Bitmap Display (size 256×256) showing multiple circles

Task 1. Drawing Circles with Floating Point Lookup

The first approach is based on Equation 1. The points x, y on the circumference of a circle at position (x_c, y_c) can be described by

$$\begin{pmatrix} x \\ y \end{pmatrix} = radius \times \begin{pmatrix} cos(\alpha) \\ sin(\alpha) \end{pmatrix} + \begin{pmatrix} x_c \\ y_c \end{pmatrix}, where\ 0° \leq \alpha < 360°. \quad (1)$$

The function `circle_float` draws a circle with a center at position x_c, y_c. The color is defined by the input argument *color* encoded as RGB value, where color[7:0] is the blue, color[15:8] the green, and color[23:16] the red part. E.g. $0xFF0000$ encodes red, $0x00FF00$ green, etc. For each α four pixels are drawn using the `draw_pixel` function.

```
1  circle_float(int x_c, int y_c, int radius, int color){
2      for (int α = 0; α ≤ 90;α++){
3          float x = radius × cos(α);
4          float y = radius × sin(α);
5          draw_pixel(x_c + x, y_c + y, color);
6          draw_pixel(x_c − x, y_c + y, color);
7          draw_pixel(x_c + x, y_c − y, color);
8          draw_pixel(x_c − x, y_c − y, color);
9      }
10 }
```

1. Implement a function `sin`. Use the file `sinlut.asm` that contains the binary representation of the values of $sin(0)...sin(90)$ as IEEE single-precision floating point numbers. Add your code into `circle_float.asm`.

2. Implement the function `cos`. Here you can use the property $cos(\alpha) = sin(90 - \alpha)$. Add your code into `circle_float.asm`.

3. Implement the function `draw_pixel`. Use the file `draw_pixel.asm` for this purpose. (You can use the plot function from the Mandelbrot exercise to get started).

4. Implement the function `circle_float`.

5. Use the RARS Bitmap Display (size 256×256) to draw multiple circles with a color and radius of your choice. Add the code to `main_float`.

6. Include a screenshot of your display to the report. Figure 1 on the previous page shows an example.

Task 2. Drawing Circle with Integer Algorithm

The following algorithms draws circles by only using integer operations. Implement a program using the RV32I instruction set that realizes the algorithm from the pseudo code below. Given algorithm:

```
circle_int(int x_c, int y_c, int radius, int color){
    d = -radius;
    x = r;
    y = 0;
    while(y < x){
        d = d + 2×y + 1;
        y = y + 1;
        if(d > 0){
            d = d - 2×x + 2;
            x = x - 1;
        }
        draw_pixel(x_c + x, y_c + y, color);
        draw_pixel(x_c - x, y_c + y, color);
        draw_pixel(x_c - x, y_c - y, color);
        draw_pixel(x_c + x, y_c - y, color);
        draw_pixel(x_c + y, y_c + x, color);
        draw_pixel(x_c - y, y_c + x, color);
        draw_pixel(x_c - y, y_c - x, color);
        draw_pixel(x_c + y, y_c - x, color);
    }
}
```

1. Implement `circle_int`. The usage of the input arguments is equivalent to `circle_float`. Add your code into `circle_int.asm`.
2. Use th RARS Bitmap Display (size 256×256) to draw multiple circles with a color and radius of your choice. Add the code to `main_int`.
3. Include a screenshot of your display to the report.

Task 3. Evaluation

1. Execute the unit test files `test_circle_int` and `test_circle_float`. Add the number of cycles required in the red fields of the evaluation spreadsheet `evaluation.xlsx`.
2. Add a data cache with a size of ≤ 512 *Bytes* to your processor by configuring the RARS data cache simulator tool. Run both: `test_circle_int` and `test_circle_float` and justify the selection of your cache parameters and fill out the red fields in the evaluation spreadsheet `evaluation.xlsx`.