# Exercise Sheet 3: Graphics Programming

The goal of this exercise is to learn to how to implement programs in RISC-V assembly to draw single pixels, rectangle and circles, as well as how to use the display and input devices of the RARS and FPGRARS simulators. Watch this video `https://youtu.be/ZCmakU874no` to see the final results.

Before starting call<br>
**cesp_setup** ↩<br>
Material for this exercise is in $HOME/cesp_course/exercises/02_graphics of CESP VM.<br>
Hint: use the units test provided to check your results.

**Task 1.** Draw rectangles

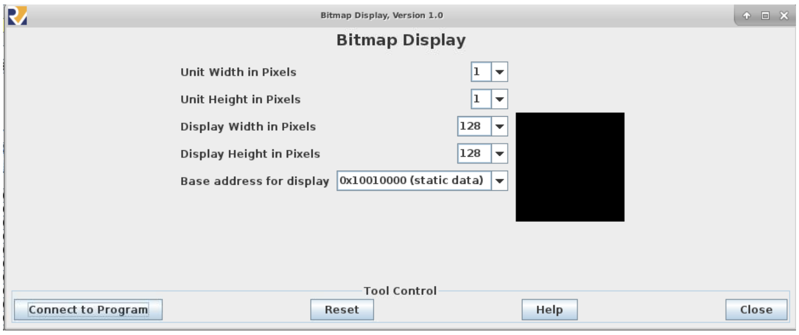
The program code of Listing 1 can draw a rectangle of user-defined size and color.

1. Implement the functions `draw_pixel` and `draw_rectangle` in RISC-V RV32IM assembly language following the STEPS in rectangle.asm and draw_pixel.asm Hint: A multiplication can be done with the `mul dst, src0, src1` instruction (we will discuss this instruction in the next chapter).

2. Use `unittest_drawpixel.asm` and `unittest_drawrectangle.asm` to execute the functions implemented in the previous step.

3. Use `unittest_drawpixel.json` and `unittest_drawrectangle.json` to verify the results.

4. To see the result, select from the RARS context menu: **Tools->Bitmap Display**. Select the values for width, height, etc. as shown below.



Bitmap Display

Unit Width in Pixels: 1

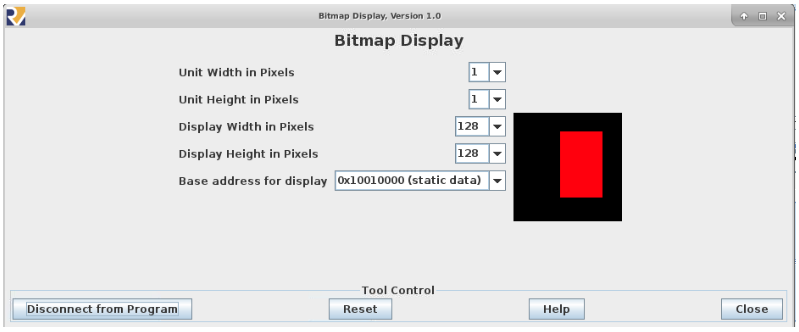
Unit Height in Pixels: 1

Display Width in Pixels: 128

Display Height in Pixels: 128

Base address for display: 0x10010000 (static data)

5. Before running the code, click **reset** and **connect to program**.



Unit Width in Pixels 1
Unit Height in Pixels 1
Display Width in Pixels 128
Display Height in Pixels 128
Base address for display 0x10010000 (static data)

6. When executing the program, a red rectangle appears on the Bitmap Display of your RARS simulator.

```
#define DISPLAY_WIDTH 256
#define DISPLAY_HEIGHT 256
#define DISPLAY_ADDRESS 0x10010000

void draw_rectangle(int x0, int y0, int x1, int y1, int color){
  // x0, y0 : coordiante of top-left corner of rectangle
  // x1, y1 : coordinate of bottom-right corner of rectangle
  // color: color in RGB format

  for(int y=y0;y<y1;y++){
    for(int x=x0;x<x1;x++){
      draw_pixel(x, y, color);
    }
  }
}

void draw_pixel(int x, int y, int color){
  int* crtaddress = (int*) DISPLAY_ADDRESS + x + y*DISPLAY_WIDTH;
  *crtaddress = color;
}

int main(){
  int fillcolor = 0xFF0000; \\RGB: red: 255, green: 0, blue: 0
  draw_rectangle(45, 45, 77, 88, fillcolor);
}
```

Listing 1: Drawing a rectangle.

**Task 2.** Draw Circles

The algorithm in Listing 2 draws circles by only using integer operations. Implement a program using the RV32IM instruction set that realizes the algorithm from the pseudo code below.

1. Implement `draw_circle`. Add your code into `draw_circle.asm`.
2. Use `unittest_drawcircle.asm` to execute the functions implemented in the previous step.
3. Use `unittest_drawcircle.json` to verify the results.
4. 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`.

```
//.include "cesplib\_rars.asm"
#define DISPLAY_WIDTH 256
#define DISPLAY_HEIGHT 256
#define DISPLAY_ADDRESS 0x10010000

void draw_circle(int x_c, int y_c, int radius, int color){
  // x_c, y_c : coordiante of center
  // radius : radius of circle in pixel
  // color: color in RGB format
  d = -radius;
  x = radius;
  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);
  }
}

int main(){
  int fillcolor = 0xFF0000; \\RGB: red: 255, green: 0, blue: 0
  draw_circle(45, 45, fillcolor);
}
```

Listing 2: Drawing a circle.

**Task 3.** Use the keyboard to interact with your program

RARS offers a memory-mapped interface to capture key strokes of your keyboard. The following video shows the functionality: RARS MMIO Keyboard [`https://youtu.be/-h3eH4ubuno`].

When a key stroke is detected, the value of address $0xFF0010000$ is set to 1. The $ASCII$ value of the key that was pressed is then saved in memory at address $0xFF0010004$. Important: You must reset $0xFF0010000$ to 0 again to be able to catch the next key stroke.

1. Implement the program code from Listing 3.
2. Check the functionality using the RARS MMIO Keyboard.

```c
#define KEYBOARD_ADDRESS 0xFF0010000
#define RED 0xFF0000
#define BLACK 0x000000

int main(){
  int x, y = 64;
  int size = 4;

  int* kb_base = (int*) KEYBOARD_ADDRESS;

  while(true){
    int key_pressed = *kb_base;
    int key_code = *(kb_base+4);

    if (key_pressed == 1){
      switch (key_code){
        case 'w':
          y -=1;
        case 's':
          y +=1;
        case 'a':
          x -=1;
        case 'd':
          x +=1;
        }
        draw_rectangle(x, y, x+size, y+size, RED);
        cesp_sleep(20); // see lecture slides or file cesplib_rars.asm
        draw_rectangle(x, y, x+size, y+size, BLACK);
      }
      key_pressed = 0x0; // Reset key vector again
    }
}
```

Listing 3: Keyboard interaction

**Task 4.** Use FPGRARS to run your code from Task 3.

Hint: RARS and FPGRARS have different addresses for display and keyboard. Use `cesplib_fpgrars.asm` to replace `cesplib_rars`.

Command to run an assembly program with FPGRARS:

```
fpgrars interaction.asm ↩
```