Exercise 6: Arithmetics with RISC-V

Integer Multiplication

Task 1. Preparation

Multiply -15×13 as binary using the algorithm shown in the lecture. You have to implement it in the next Task as assembly code.

Task 2. Multiplication

Implement the algorithm for integer multiplication given by the C-like pseudcode below using the RV-32I ISA (using the `mul` instruction from RV-32M is not allowed). **Important: Do not use Compiler Explorer here.**

```
int multiply(int a, int b){
    int result = 0;
    int factor = b;

    for (int i=0; i< sizeof(int)*8; i++){
        int last_binary = factor & 0x1;
        if (last_binary == 1){
            result = result + (a << i);
        }
        factor = factor >> 1;
    }
    return result;
}

int main(){
    int a = 15;
    int b = 13;
    int c = multiply(a,b);
    printf("%d * %d = %d\n", a, b, c);
    return 0;
}
```

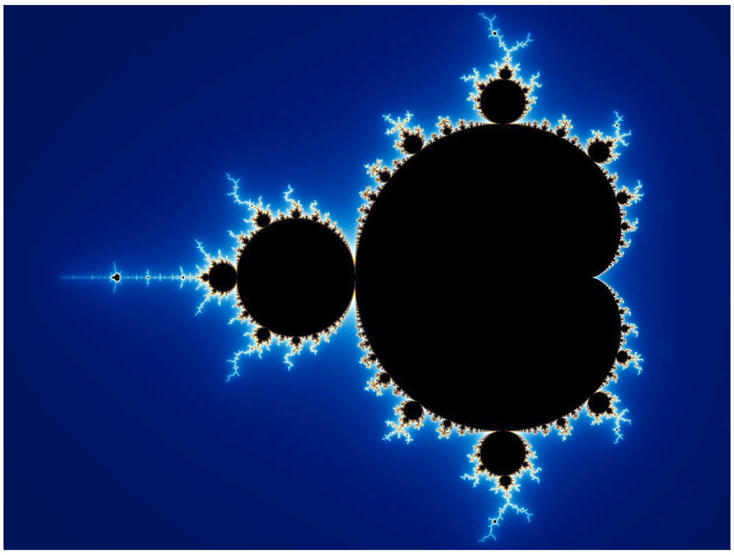
Test your code by writing a unit test for the following input values:

- 2×2,
- 45×32,
- -2×3,
- -4×7,
- -5×-6.

Further instructions are given in the STEPS in multiplication.asm.

Fixed and Floating Point Calculation

In this exercise you will implement the algorithm for drawing the Mandelbrot set that is shown below.



The Mandelbrot set is a set of complex numbers for which the function $f_c(z) = z^2 + c$ does not diverge when starting from $z = 0$. We will use the iterative formula in this exercise:

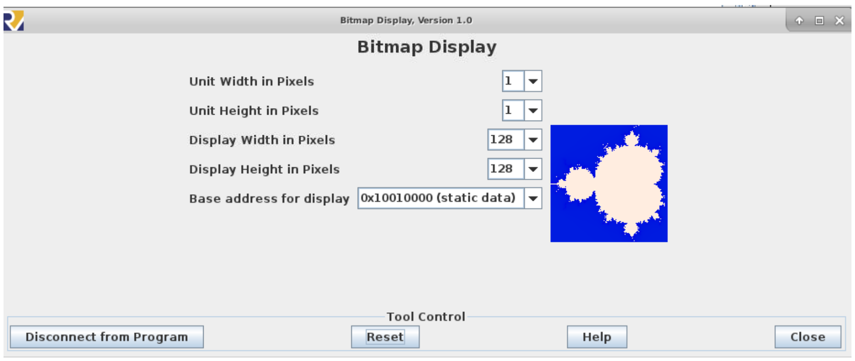
$$z_{n+1} = z_n^2 + c, c \in \mathbb{C} \tag{1}$$

Task 3. Fixed point implementation
At first we will implement the algorithm with the `Q24.8` fixed point format. You can realize operations with `Q24.8` using RV32-IM assembly instructions by combining instructions.

Example:

```
fixed_q24.8 a = 5.5 = 0x00000580
fixed_q24.8 b = 1.25 = 0x00000140
a + b = 6.75 = 0x6c0
a * b = 6.875  = (0x580 * 0x140) >> 8 = 0x6E0
```

Implement the algorithm based on the C-like pseudo-code from Listing 1 using the `Q24.8` fixed point format. The result in RARS looks like this:



Bitmap Display, Version 1.0

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)

Steps:

1. Use the C-like pseudocode mandelbrot_start.c to get started and use Compiler Explorer
2. Implement a function in mandelbrot_start.c to perform fixed-point multiplication
3. Replace all the constant (e.g. 2.0, -1.5) with the correct values as hexadecimal integer (e.g. 2.0 → 0x200)
4. Update mandelbrot_start.c: Use the function from step 2 for all multiplications of TWO fixed_q24.8 numbers. Multiplying ONE fixed_q24.8 with an integer can be done as before.
5. Copy the assembly output to RARS.
   Remember: When copying assembly code from Compiler Explorer to RARS, you need to do things that are normally done by a linker, e.g. allocating memory for the display, checking where static variables are stored ... (this highly depends on your implementation in C).
6. Run the program with RARS until you get the expected result
7. (Optional) Run the program with FPGRARS. Remember that you need to update at least the DISPLAY_ADDRESS address

Task 4. Floating Point Implementation

Copy your assembly code into a new file and update it to use the IEEE Floating-Point Format with single precision, instead of fixed point. In the pseudo-code on the next page every `fixed_q24.8` keyword, then becomes `float`. Use the RV32F instructions and floating point registers to realize the program.

Important: Do not use Compiler Explorer here.

```c
#define IMAGE_WIDTH 128
#define IMAGE_HEIGHT 128
#define MAX_ITERATIONS 50
#define DISPLAY_BASE 0x10010000



void plot(int x, int y, int iterations){
  int* crt_address = (int*) DISPLAY_BASE + y * IMAGE_WIDTH + x;
  int color = 0xFF - iterations * 8;
  *crt_address = color;
}

void mandelbrot(fixed_q24.8 x_start, fixed_q24.8 y_start, fixed_q24.8
   x_stretch, fixed_q24.8 y_stretch){
  fixed_q24.8 Zr, Zi, Cr, Ci, Tr, Ti;
  fixed_q24.8 two = 2.0;
  fixed_q24.8 four = 4.0;
  for(int y=0; y<IMAGE_HEIGHT; ++y){
    for(int x=0; x<IMAGE_WIDTH; ++x){
      Zr = Zi = Tr = Ti = 0.0;
      Cr = (x_stretch * x / IMAGE_WIDTH + x_start);
      Ci = (y_stretch * y / IMAGE_HEIGHT + y_start);
      int iterations = 0;
      for (iterations = 0; iterations < MAX_ITERATIONS && (Tr+Ti <= four);
          ++iterations){
        Zi = two * Zr * Zi + Ci;
        Zr = Tr - Ti + Cr;
        Tr = Zr * Zr;
        Ti = Zi * Zi;
      }
      plot(x, y, iterations);
    }
  }
}


int main(){
  fixed_q24.8 x_start = -1.5;
  fixed_q24.8 y_start = -1.0;
  fixed_q24.8 x_stretch = 2.0;
  fixed_q24.8 y_stretch = 2.0;

  mandelbrot(x_start, y_start, x_stretch, y_stretch);
  return 0;
}
```

Listing 1: Function to draw the Mandelbrot Set.