**Specification**

You will need to implement a set of specific subroutines indicated in these lab instructions. You are required to start with the skeleton code provided ([lab5\_s20\_template.asm](https://drive.google.com/open?id=1ouGrq4X9G0OtT-lwono1jTXw0OCq113v)) and **may not change the function names or arguments at all.** Please rename the file to Lab5.asm and start with it. To receive any credit for your subroutines, **Lab5.asm must assemble** both **on its own** and with the test file. On its own, the template file shouldn’t print or draw anything -- it is just a set of subroutines.

A test file ([lab5\_s20\_test.asm](https://drive.google.com/open?id=1lbJugCqef49AFRF8zYlkRau8PZWQfjXy)) tests each one of your subroutines and includes (at the very end) your subroutines from Lab5.asm (based on the above template file). You should modify the test to include Lab5.asm instead of lab5\_s20\_template.asm. Don’t put the functions in this the test, they go in Lab5.asm. **We will not use your test file!** In order for your subroutines to function properly, **you must use** the instructions **JAL and JR** to enter and exit subroutines. You must save and restore registers as required in MIPS. Our test file will look very much like this one, so you should ensure that your functions work with it!

*Functionality*

The functionality of your program will support the following:

1. All pixels should be in the range x in [0,128) and y in [0,128) (the parenthesis means not including 128).
2. Pixels start from (0,0) in the upper left to (127,127) in the lower right.
3. Pixel values are referenced in a single word using the upper and lower half of the word. So, for example, 0x00XX00YY) where XX and YY can be 0x00 to 0x7F.
4. All colors should be RGB using a single 32-bit word where the top byte is zero. So, for example, 0x00RRGGBB where RR, GG, and BB can be 0x00 to 0xFF.
5. Clear the entire bitmap display to a color c.
6. Draw a circle with center at (xc, yc) and radius r filled of a given color c.
7. Draw a circumference with center at (xc, yc) and radius r of a given color c.

*Macro Descriptions*

You are required to use these macro definitions without modification. These macros should be in the Lab5.asm file. You may use additional macros if you like but be sure to include them in Lab5.asm.

**getCoordinates(%input %x %y):** Macro that takes as input coordinates in the format (0x00XX00YY) and returns 0x000000XX in %x and returns 0x000000YY in %y. Do not use any registers other than the input registers to write this macro.

**formatCoordinates(%output %x %y):** Macro that takes Coordinates in (%x,%y) where %x = 0x000000XX and %y= 0x000000YY and returns %output = (0x00XX00YY). Do not use any registers other than the input registers to write this macro.

**push(%reg):** Macro that stores the value in %reg on the stack and moves the stack pointer. The only register that is altered in this macro is $sp.

**pop(%reg):** Macro takes the value on the top of the stack and loads it into %reg then moves the stack pointer. The only registers altered are %reg and $sp.

*Subroutine Descriptions*

These subroutines should be in the Lab5.asm file. You may use additional functions if you like, but they should be included in Lab5.asm as well.

It is important that these subroutines do **NOT** display any text to the screen using syscalls. If so, this will interfere with the grading script and result in point deductions. **You may print strings and characters in the lab5\_s20\_test.asm file, but not in Lab5.asm!**

We recommend that you try to implement these functions in roughly the given order. This order “builds up” so you get an understanding of memory-mapped IO, the bitmap display, and how functions work.

**clear\_bitmap:** Given a color, this function will fill the bitmap display with that color. **It is not required that this call any other functions, but you may want to use draw\_pixel.**

**Inputs:**

    $a0 = Color

**Outputs:**

    No register outputs

**Side-Effects**:

    Colors the Bitmap display (all RGB pixels from 0xFFFF0000 to 0xFFFFFFFC) all the same color. (Question for yourself, why 0xFFFFFFFC and not 0xFFFFFFFF?)

**draw\_pixel:** Given a coordinate in $a0, this function will color a pixel in the image according to the RGB value given by register $a1. This works by storing the RGB value in the appropriate location of the row-major bitmap array starting at address 0xFFFF0000.

You should do some error checking to ensure the pixel is within range. If the XX or YY values “overflow” and are more than 8-bits, you could have segmentation fault errors when storing to the memory-map. We will not be grading this error checking, but **it will save you time debugging**!

**Inputs:**

$a0 = coordinates of pixel in format (0x00XX00YY)

$a1 = color of bitmap in format (0x00RRGGBB)

**Outputs:**

No register outputs

**Side-effects:**

Draws a pixel in the Bitmap Display

**get\_pixel:** Given a coordinate, returns the color of that pixel. This is used for some “spot checks” in our test code.

**Inputs:**

$a0 = coordinates of pixel in format (0x00XX00YY)

**Outputs:**

$v0 = color of the pixel at that coordinate in format (0x00RRGGBB)

**Side-effects:**

None

**draw\_solid\_circle:** Given the coordinates of the center (xc, yc) and the radius r draws a filled circle of a desired color. The algorithm (given bellow) test each point of the circumscribed square and plot it if the point lies **inside** the circle

(x – xc)^2 + (y – yc)^2 = r^2. **This function should use draw\_pixel.**

**Inputs:**

$a0 = coordinates of the circle center in format (0x00XX00YY)

$a1 = radius of the circle

$a2 = color in format (0x00RRGGBB)

**Outputs:**

No register Outputs

**Side-effects:**

Draws a filled circle in the Bitmap Display.

draw\_solid\_circle(int xc, int yc, int r)

    xmin = xc-r

    xmax = xc+r

    ymin = yc-r

    ymax = yc+r

    for (i = xmin; i <= xmax; i++)

        for (j = ymin; j <= ymax; j++)

            a = (i – xc)\*(i – xc) + (j – yc)\*(j – yc)

            if (a < r\*r )

                draw\_pixel(x,y)

A couple notes:

1. You should accurately follow this algorithm and not “improve” it at all. If you do, you may introduce pixel errors that can cause point deductions.
2. You can assume that the center coordinates and radius are “correct” (in bounds) and do not have to do error checking.
3. Pay attention to the inequalities (for example, <= is not the same as <).
4. This computation should not use floating point numbers. It is all two’s complement (signed) integers.

**draw\_solid\_circle:** Given the coordinates of the center (xc, yc) and the radius r, uses draw\_circle\_pixels and Bresenham’s circle drawing algorithm (given below), to draw the circumference with the specified color line. **This function should use draw\_circle\_pixels.**

**Inputs:**

$a0 = coordinates of circle center in format (0x00XX00YY)

$a1 = radius of the circle

$a2 = color of line format (0x00RRGGBB)

**Outputs:**

No register Outputs

**Side-effects:**

Draws a circumference in the Bitmap Display.

draw\_circle(xc, yc, r)

    x = 0

    y = r

    d = 3 - 2 \* r

    draw\_circle\_pixels(xc, yc, x, y)

    while (y >= x)

        x=x+1

        if (d > 0)

            y=y-1

            d = d + 4 \* (x - y) + 10

        else

            d = d + 4 \* x + 6

        draw\_circle\_pixels(xc, yc, x, y)

A couple notes:

1. You should accurately follow this algorithm and not “improve” it at all. If you do, you may introduce pixel errors that can cause point deductions.
2. You can assume that the center coordinates and radius are “correct” (in bounds) and do not have to do error checking.
3. Pay attention to the inequalities (for example, >= is not the same as >).
4. This computation should not use floating point numbers. It is all two’s complement (signed) integers.
5. Remember that multiplication by a power of 2 can be done with a shift operation.

**draw\_circle\_pixels:** Given the coordinates of the center (xc, yc) and the (x, y) values from the Bresenham’s circle drawing algorithm, plots the circumference points. **This function should use draw\_pixel.**

**Inputs:**

$a0 = coordinates of circle center in format (0x00XX00YY)

$a1 = color in format (0x00RRGGBB)

$a2 = current x value from the Bresenham's circle algorithm

$a3 = current y value from the Bresenham's circle algorithm

**Outputs:**

No register Outputs

**Side-effects:**

Draws the points of the circumference using the circle’s symmetry.

draw\_circle\_pixels(xc, yc, x, y)

    draw\_pixel(xc+x, yc+y)

    draw\_pixel(xc-x, yc+y)

    draw\_pixel(xc+x, yc-y)

    draw\_pixel(xc-x, yc-y)

    draw\_pixel(xc+y, yc+x)

    draw\_pixel(xc-y, yc+x)

    draw\_pixel(xc+y, yc-x)

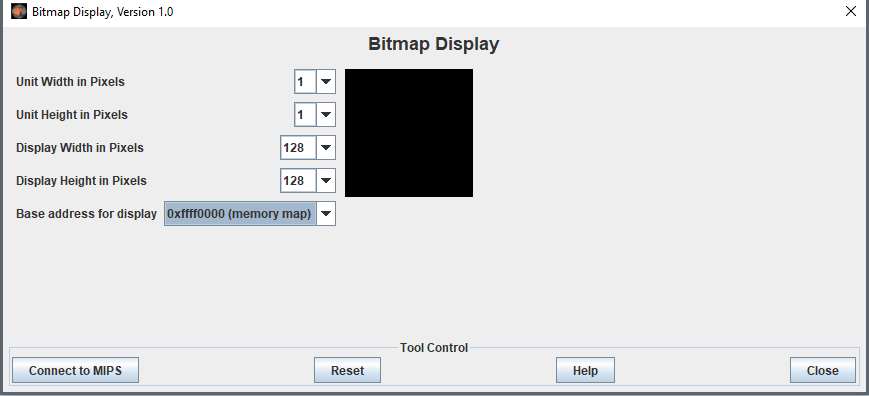
    draw\_pixel(xc-y, yc-x)

A couple notes:

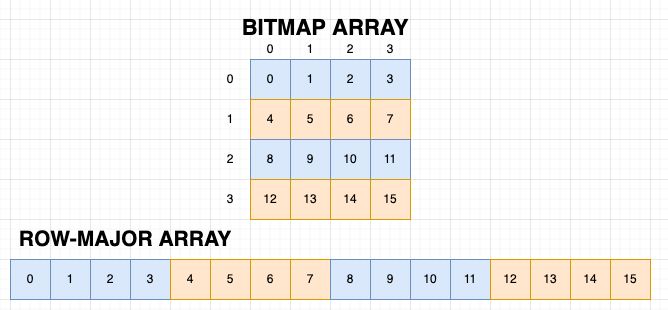
1. You should accurately follow this algorithm and not “improve” it at all. If you do, you may introduce pixel errors that can cause point deductions.
2. You can assume that the center coordinates are “correct” (in bounds) and do not have to do error checking.
3. This computation should not use floating point numbers. It is all two’s complement (signed) integers.

## *Test Output*

The test output for this lab is visual and requires you to use the MARS Bitmap Display tool (in Mars select Bitmap Display from the Tools menu). You should modify the settings of the bitmap display to be 128 x 128 pixels and to have a base address of the memory map (0xffff\_0000) as shown here:



Press “Connect to MIPS” to use this in your program.



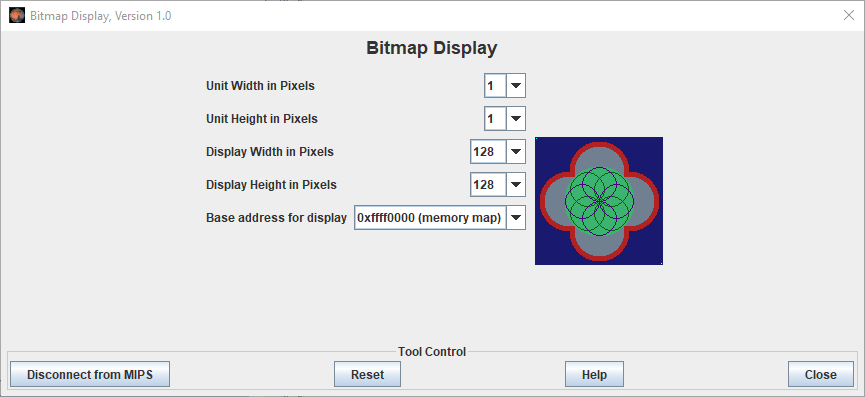
The bitmap display is a grid of 128 x 128 pixels that displays a color based off the value written to the address corresponding to that pixel. In the example above, you can see how the coordinates of the pixel relate to the array in memory for a 4 x 4 pixel bitmap. For example if you wanted to color the pixel at row 2, column 3 (2,3) you would take the base address of the of the first pixel and offset that by +11 which is (2 \* row\_size) + 3 to locate the correct pixel to color.

We will be grading your solution by dumping the memory-mapped IO segment as hexadecimal ASCII and comparing with the correct results. **You will miss all the points if you do not use the above size and base address configuration! In addition, your Lab5.asm should not display any text using syscalls as this will interfere with the grading output.** If you want, you can also display the memory-mapped segment using a command line argument like this:

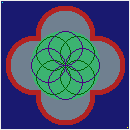
java -jar Mars4\_5.jar nc 0xffff0000-0xfffffffc lab5\_s20\_test.asm

## *Sample Input/Outputs*

You are expected to read through and understand how the provided lab5\_s20\_test.asm file works. The test file will print to the console the state of the S registers before and after calling a subroutine, provide inputs, and test certain pixels to make sure that it’s drawn in the correct place. This is what the output of your completed lab should look like:



And the details of bitmap should display the following exactly:



Please note the following:

* The pixels draw “on top of” each other so the order of the drawn shapes matters. For example, the red circles are drawn before the gray circles, so they are “under” them.

This output of the tests are available in this file [lab5\_s20\_test.hex](https://drive.google.com/open?id=1UaX6gCDY411yM1h6zWMHRMtKQRdAHt7H) if you wish to compare. You can compare files online using a “diff” utility like [Diffchecker](https://www.diffchecker.com/) or [the bash “diff” command](https://www.geeksforgeeks.org/diff-command-linux-examples/).

For full credit, **your output should match ours exactly**.

### ***Pseudocode***

You should write pseudocode that outlines each function. Your pseudocode will appear at the start of each function in Lab5.asm. Guidelines on developing pseudocode can be found here: <https://www.geeksforgeeks.org/how-to-write-a-pseudo-code/>

You may modify your pseudocode as you develop your program. **Your pseudocode must also be present in your final submission**.

## *Automation*

Note that part of our grading script is automated, so **it is imperative that your program’s output matches the specification exactly**. Output that deviates from the spec will cause point deduction.

**You should not use a label called “main” anywhere in Lab5.asm.** If you do, it will fail to work with our test cases and your assignment will not be graded.