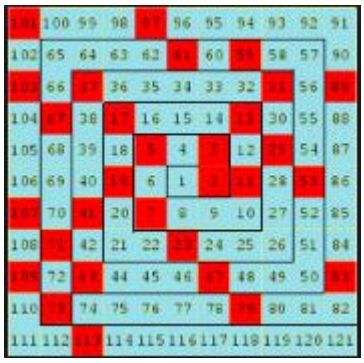
COMP30080 Processor Design

Assignment 3 Fergal Lonergan

Ulam's Spiral

The object of this assignment was to print Ulam's Spiral, a spiral which indicates prime numbers. To create the spiral you start in the centre of your display at the corresponding pixel for the value 1. You then move right, up, left and down continuously outward for as many values as you would like. You then distinguish between the prime numbers and not prime numbers by some means. In the following examples the primes are indicated in red, while the other numbers are shaded blue.



You must also increase the distance your travel in the x and y directions by 1 every time you flip directions in the corresponding planes to ensure you do not overwrite values from before. We always move to a new pixel location adjacent to the current pixel also as indicated above.

For my display the primes would be indicated by a black pixel and the non-primes by a white pixel.

MIPS program

I will give a synopsis of my program and how I implemented Ulam's spiral.

.data

Firstly I allocated space for my display. I reserved 65536 words to store my bytes for my display (256*256 display). I would then set these words equal to hex values depending on what colour they needed to be to represent prime or not. If I did not do this the subsequent memory blocks would overlap my display's memory and then unwanted values would be displayed to the bitmap. I also have a Boolean array I call Numbers that holds 62025 values. If the value in this Boolean array is set to one then the corresponding number is a prime, if it is set to 0 then the corresponding number is not prime. Furthermore I have many .eqv statements that I use to make my program more legible. I use these to hold RGB hex values for colours, start addresses for my bitmap, the max number of pixels in my display, the max number of elements in my Numbers array and the amount of bytes needed to navigate my bitmap. Finally I use .align 2 to be safe and make sure that arrays don't overlap.

More details can be found in the comments in the code.

.text

My main program is broken into 4 subroutines, Seive, LoadDisplay, ClearScreen and DisplayBitMap. I will explain each subroutine briefly. For a more detailed explanation please read the comments in the code.

More details can be found in the comments in the code.

Seive

My Seive subroutine uses the Sieve of Eratosthenes we used in assignment one and calculates all the primes between 1 and 62025. We begin with a binary array were every value is set to 1 denoting a prime. The Sieve of Eratosthenes starts at the value two. It then iterates through all the multiples of two until it reaches our boundary condition of 62025 and sets all these values to 0 denoting they are not prime. Next it iterates and moves to 3 and does the same. It checks to see if the current number is in fact a prime itself, as if it is not, all the multiples of that number will already have been set to "not prime" by its lowest factor. (ie for every number that 4 is a factor of, 2 is also a factor, and seeing as we have already checked all the multiples of two, we do not need to check the multiples of 4.)

We continue to iterate through all values from 2-251. The reason we stop at 251 is that 255 is the square root of 62025. At this point all possible combinations will have been checked by lower factors and checking multiples of numbers beyond this value is redundant. However, 255, 254, 253 and 252 are not primes, so the lowest factor of these numbers will have already set any of their multiples to a binary value of 0 denoting not prime, so we can stop our loop at 251. The sieve then stores these values in our binary array numbers to be used later.

LoadDisplay

The LoadDisplay subroutine basically loads all the necessary start addresses and loop conditions that will be needed for anything to do with the Bitmap display.

ClearScreen

This subroutine sets every pixel in my bitmap display to the RGB hex value of white (0x00FFFFFF) so that it is "cleared" to white. It is simply a loop that iterates through every pixel until they are all white.

I also added a small section to this subroutine that makes the pixel representing the integer 1 in my display blue so that it is easy to locate for testing.

DisplayBitMap

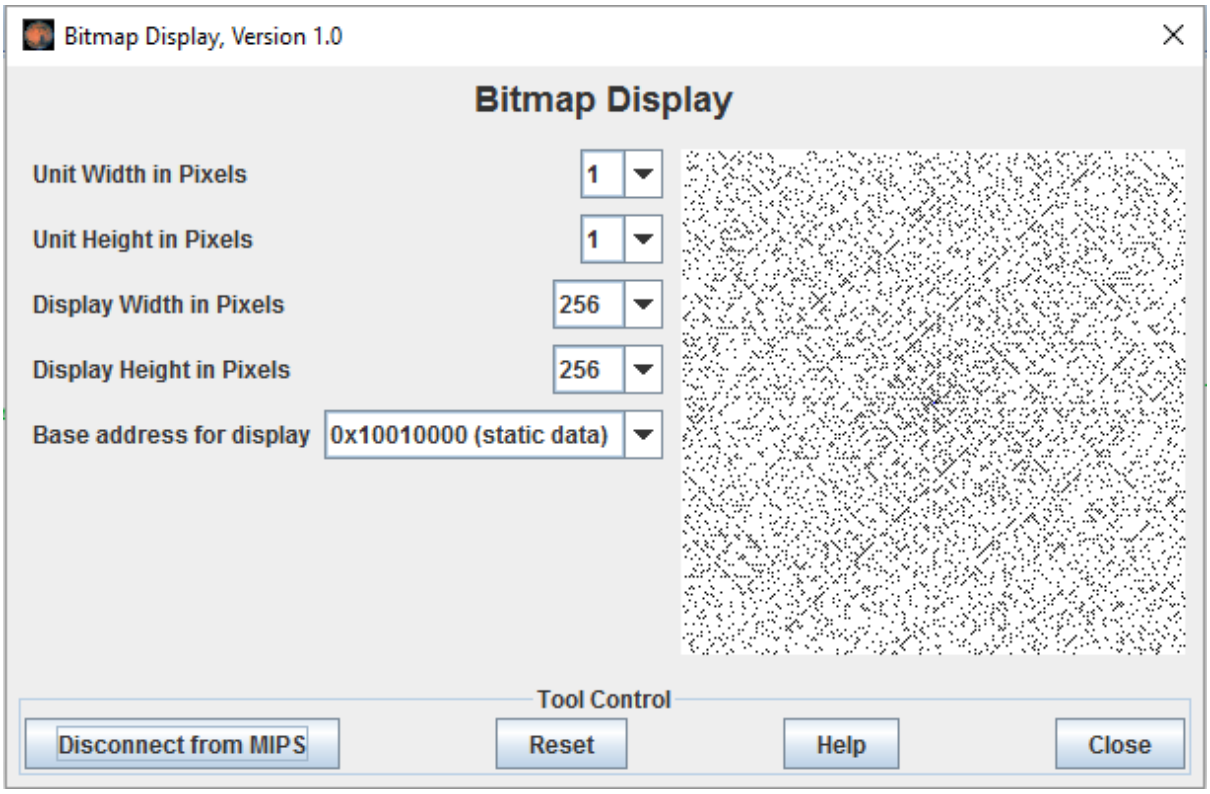
My final subroutine displays my Ulam's Spiral on my bitmap. Within it there are two subroutines XDirection and YDirection. The XDirection subroutines deals with changing any pixel values when the spiral is moving in an x direction (ie right or left). It starts by checking to see if the current element is prime or not. It changes the value of the next pixel appropriately. It checks to see if the loop conditions have been broken, if not it loops and checks to see if the next number is prime or the loop conditions have been broken.

Once the loop conditions have been met and we have travelled the specified number of pixels to the right we then increase the amount of pixels needed to be travelled in the X direction by one and multiply our shifting constant (4) by -1 so that it moves in the opposite direction (ie left) the next time the spiral moves in the x direction. By incrementing and changing direction we can ensure we check all the pixels which are to be changed by the XDirection subroutine.

We then move to the Y direction and do the same thing as in the X direction, however now to move up and down in the Y direction, our shifting constant is 1024. It starts at -1024 as we wish to move upwards first.

We have a counter that keeps track of how many pixels have been checked to date. When this is exceeded the program ends.

The following is the output of my program.



To check I zoomed in on the centre of the spiral and compared it with the spiral from the start of my report. Remembering that blue indicates the integer 1, primes are shaded black and non-primes white, and I move in a right, up, left, down manner we see the output is correct.