# Week 3 Lab A: Data representation

## Objectives

Develop understanding and experience of:

1. How the underlying representation for text and images relates to bytes
2. The bmp image format
3. ASCII as the basis for text representation

## Bits and Bytes

1. If we are using unsigned integers, what range of decimal numbers can we store in a byte?

0 to 255

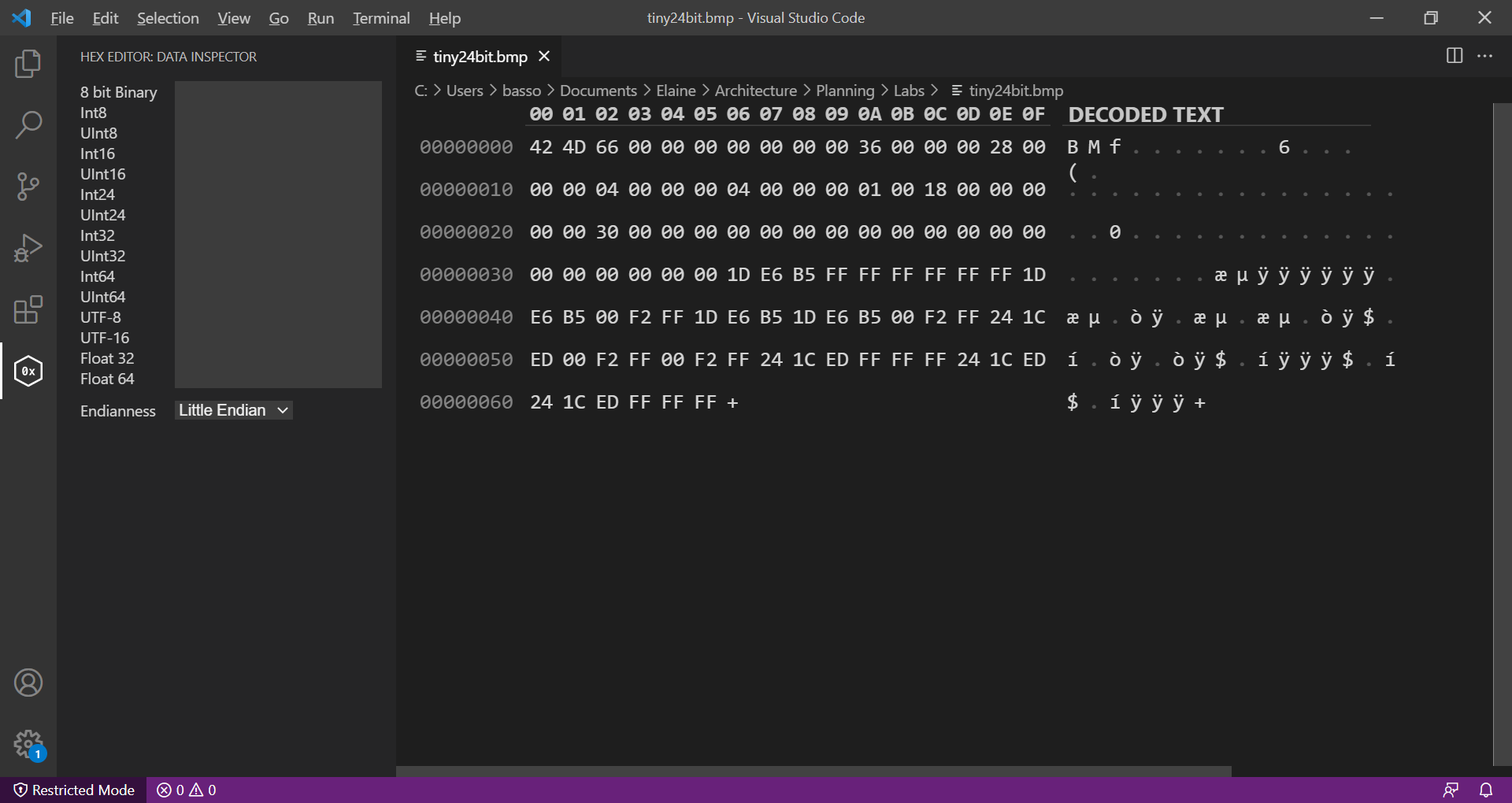
1. How many bytes do we need to store a single character in ASCII?1
2. In standard RGB notation, how many bytes do we need for each pixel? 3 bytes (one for each of red, green and blue)
3. How many bits is that? 24 bits in total
4. What is the RGB value for white (decimal and hex representation)?

Decimal (255, 255, 255)

Hexadecimal FF FF FF

## Representing images

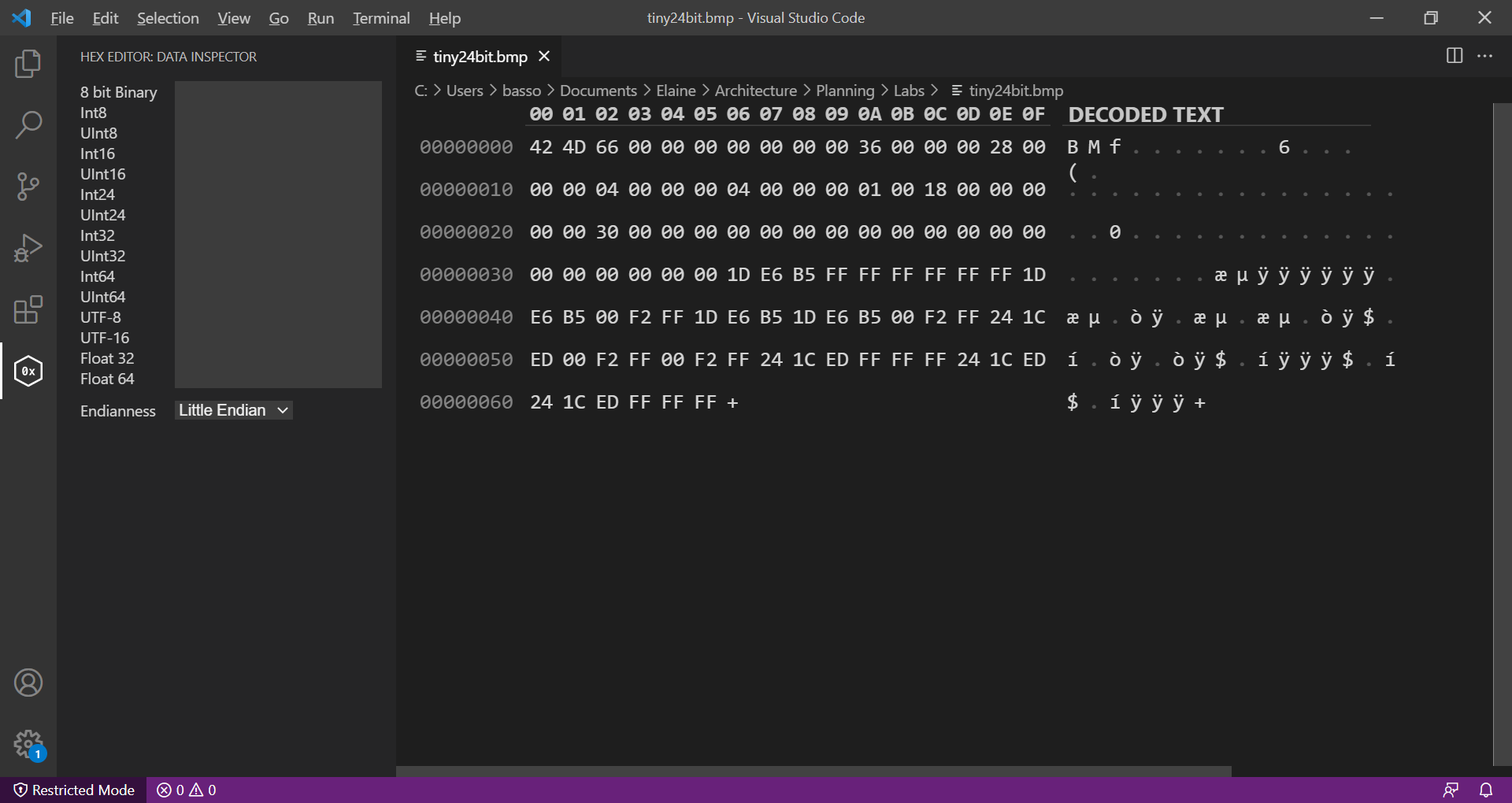
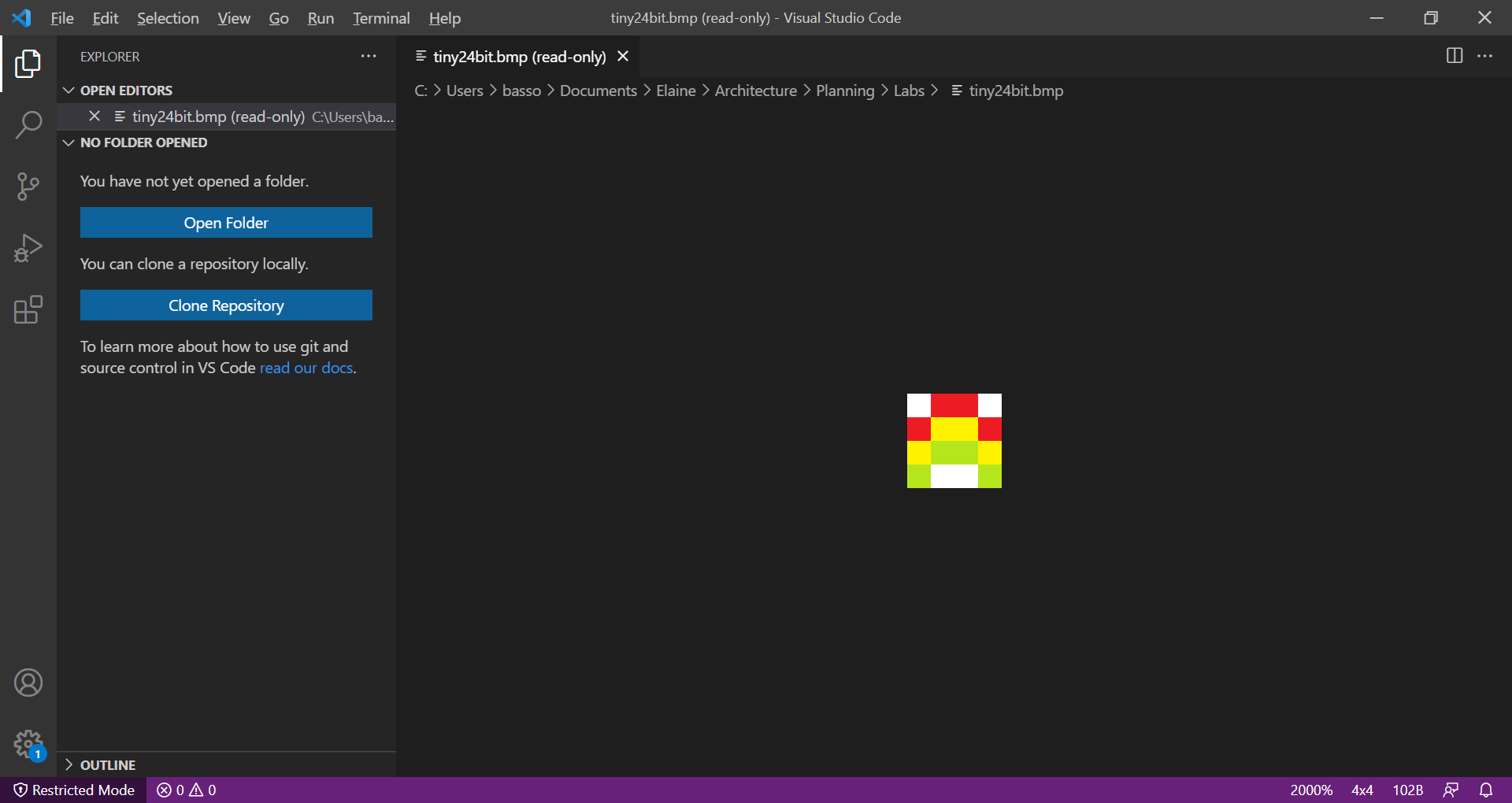
Looking at the format for the data for a bmp image.



Width of image

Height of image

Number of bits per pixel

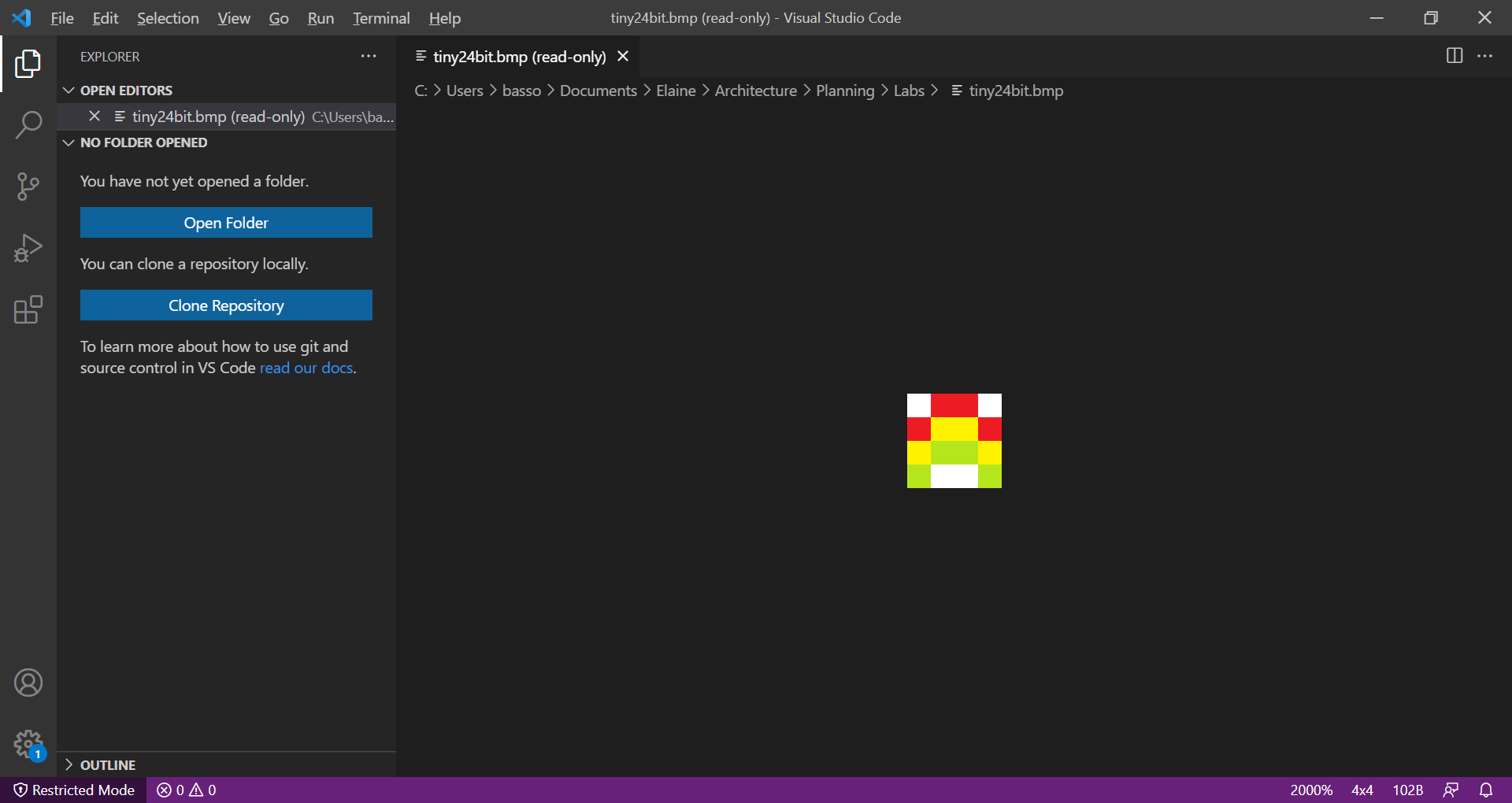


Number of bytes of pixel data

End of metadata

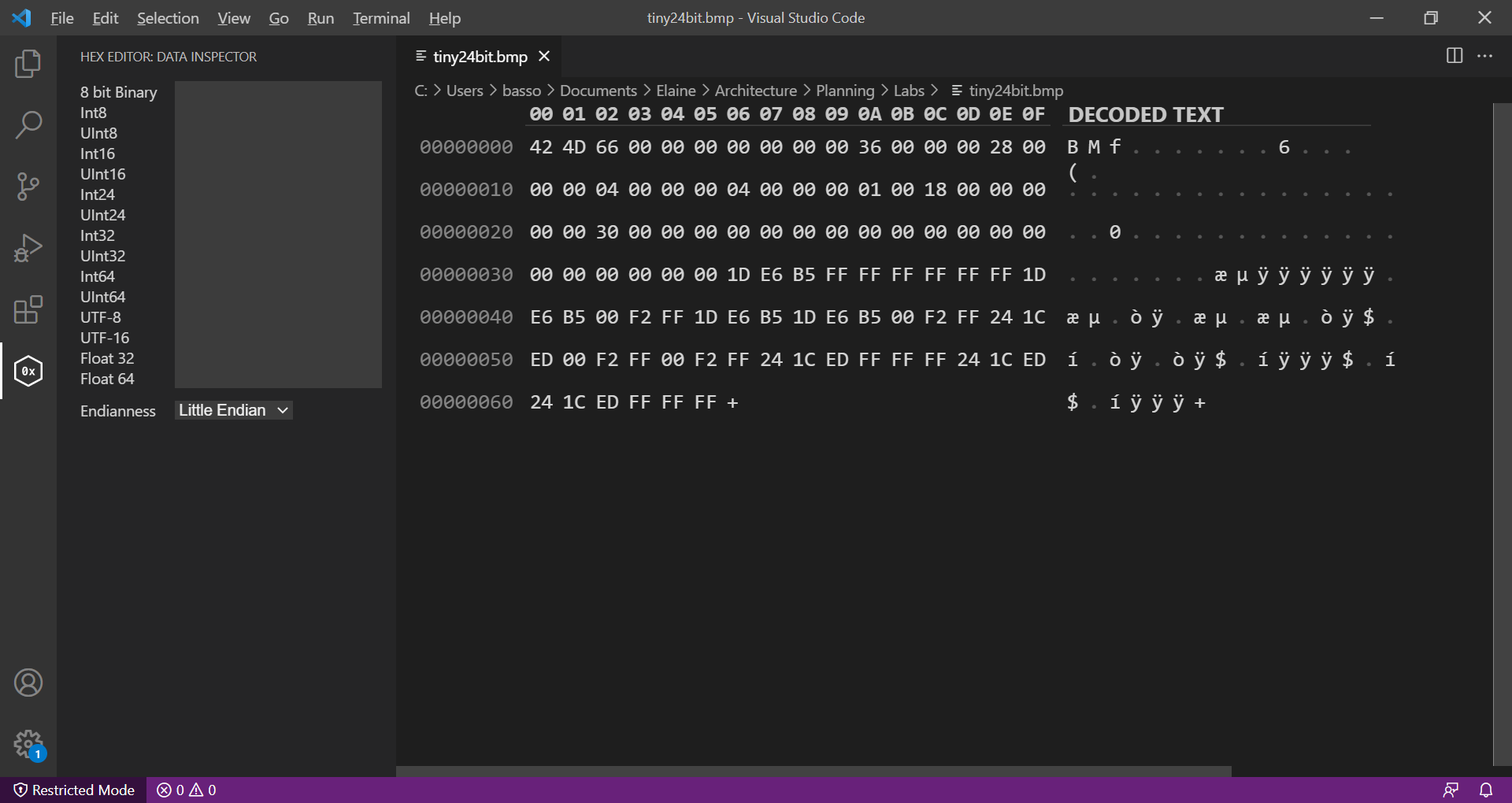
File Size

Position of start of pixel array

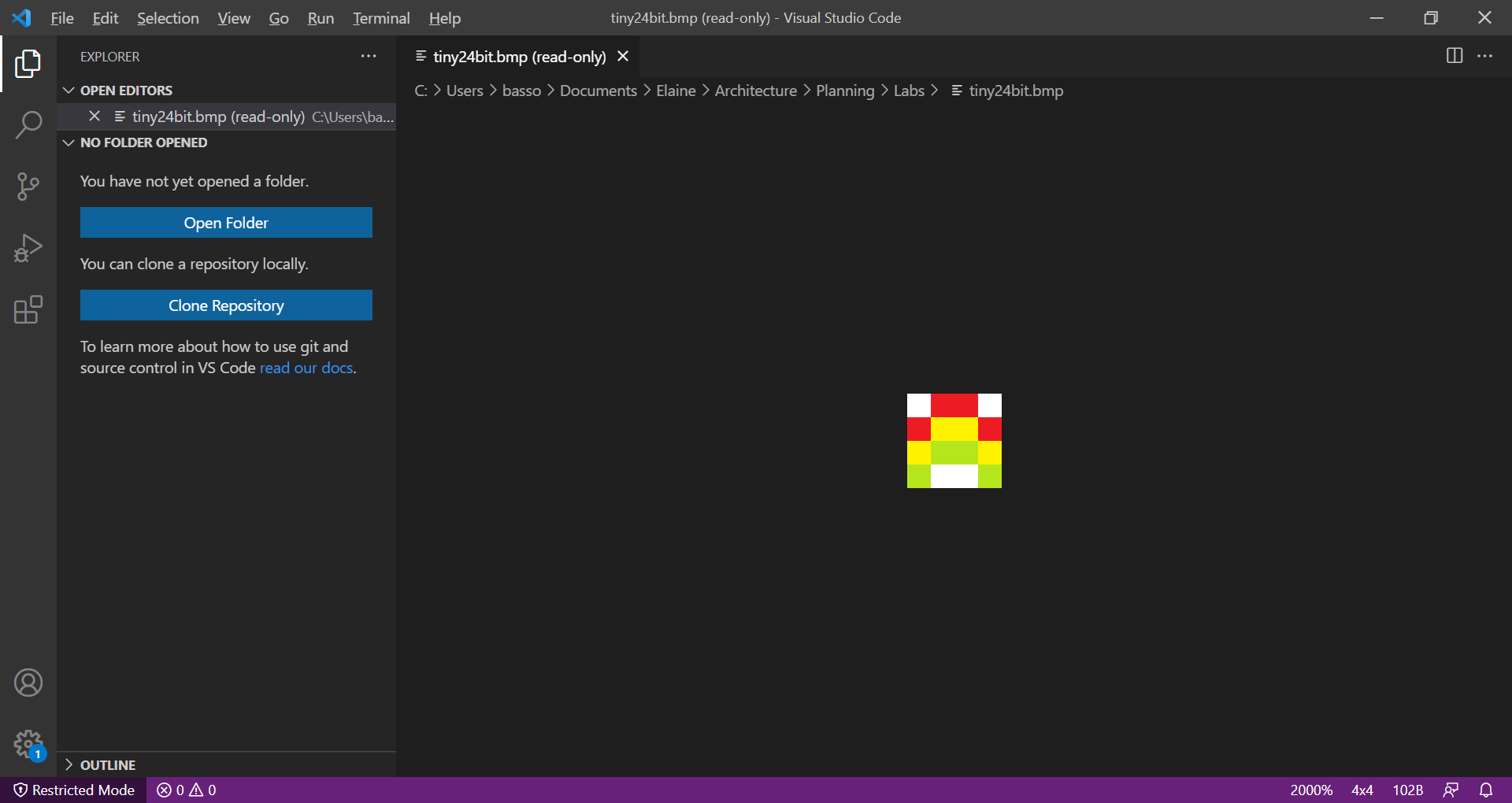


The metadata contains general information about the image file. The pixel data starts from the bottom left pixel of the image. It continues along the bottom row, then up a row.

In bmp file format Colours are represented as BGR (Blue, Green, Red) rather than the more usual RGB. It is still RGB, but the system is little-endian, so the last value (B) comes first. There’s more explanation of little-endian in the extension.



Colour data for first three pixels



1. Looking at images in bmp format, follow the next part along with the tutor
2. Download the file **tiny\_24\_bit.bmp** from the folder for lab part A on Moodle and open in Visual Studio Code.
3. VS Code should open it as an image preview, but it will be tiny as it is a 4x4 pixel image. You should be able to zoom up to at least 1000%. In the bottom bar of VS Code you can click on the zoom level which might say “Whole Image”.
4. Right-click on the image name at the top of the tab and choose “Reopen Editor With…”
5. Choose Hex Editor. If hex editor does not come up on the list, start typing hex editor. If it still doesn’t come up, you might need to install the extension.
6. The first part of the file is metadata, the pixel data starts at position X’36’. We will look at the data together. You should be used to RGB colours with each having a value between 0 and 255. In bmp file encoding, the order of the colours is BGR, that is blue, green, red.
7. Your challenge is to change the colours in the image by changing the hex values only. You should **change all the green parts of the rainbow to be blue** instead. Save the file and reopen with the Image preview to check your changes.
8. Add images to this report to show your changed hex values and the changed image.

A blue colour will have the blue value from RGB much higher than the other two. I will choose RGB A1 A0 EE. I need to reverse the order to EE A0 A1. The green pixels were 1D E6 B5 in the file so I replace all of those with my new value EE A0 A1

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

## Representing text

1. Looking at the ASCII table, there is an Excel version of the ASCII table in the lab files on Moodle
2. Open Visual Studio Code (VS Code)
3. Choose file and new file
4. Ignore any message that comes up and type in a short message (more than one word) using standard English characters.
5. Save the file as plain text (file extension .txt)
6. Right-click on the filename at the top of the tab you are using and open with the hex editor.
7. Look at the hex version of the message and add an image to this report.
8. Use the ASCII table to change the case of the message by altering the hex digits. That is, change lower to upper case and vice-versa. Use the ASCII table to work out what change to make. Take an image of the hex editor and add it to this report.
9. Write down a general rule for changing lower case to upper case using ASCII values.

To go from lower to upper case you would subtract hex 20 (decimal 32) from the value.

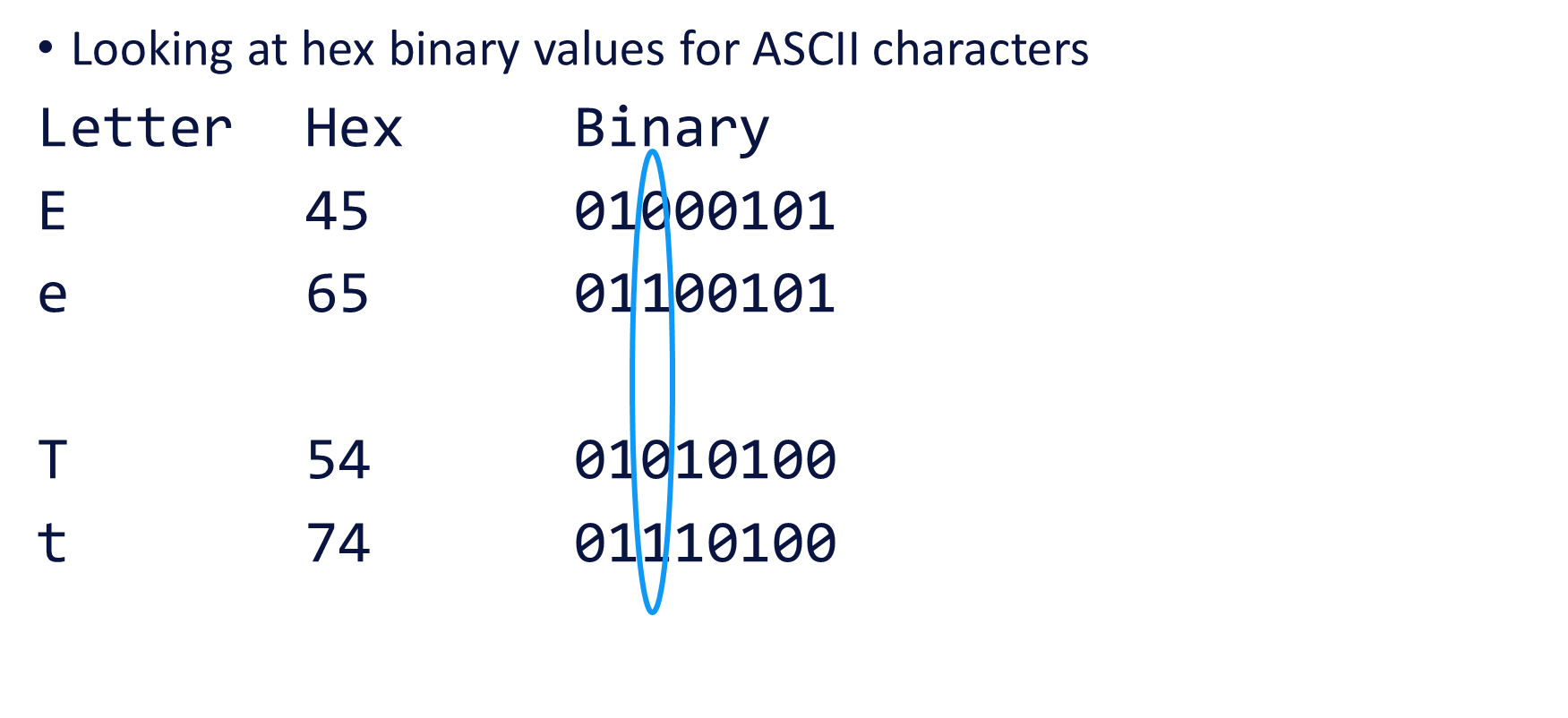
## Extension Exercises

There are extension exercises on text representation and image representation. The exercise on colour depth includes more background information on bmp file formats and is designed to be challenging but interesting and is best approached by discussing in a small group.

### Data representation

The first exercise asked you to change the case of a message by manipulating the hex digits of the ASCII representation of the message. Now you should look at how logic gates could be used to change the case.

Create a circuit that has an input as an 8-bit ASCII representation of a character, you should assume that it will always be a letter of the alphabet (upper or lower case). The output of your circuit should represent the lower case letter (that is converting upper case to lower case and leaving it the same if it was already lower case). Use logic gate(s)s, wires and constants as necessary to achieve this. Hint: You should compare the upper and lower case representation of some letters in binary to see where the difference is and consider how you could use bitwise operations.



The only digit that changes is in the position shown. That digit needs to be 1 for the lower case character.

We need the output to be 1 in all the positions where it was a 1 already and ensure that the specific position is always 1. We can use a bitwise OR with 00100000.

To specify a constant in Logisim evolution, we need a hex value

Binary 0010 0000

Hex 2 0

A picture containing diagram

Description automatically generated

A picture containing diagram

Description automatically generated

There are other ways to achieve this.

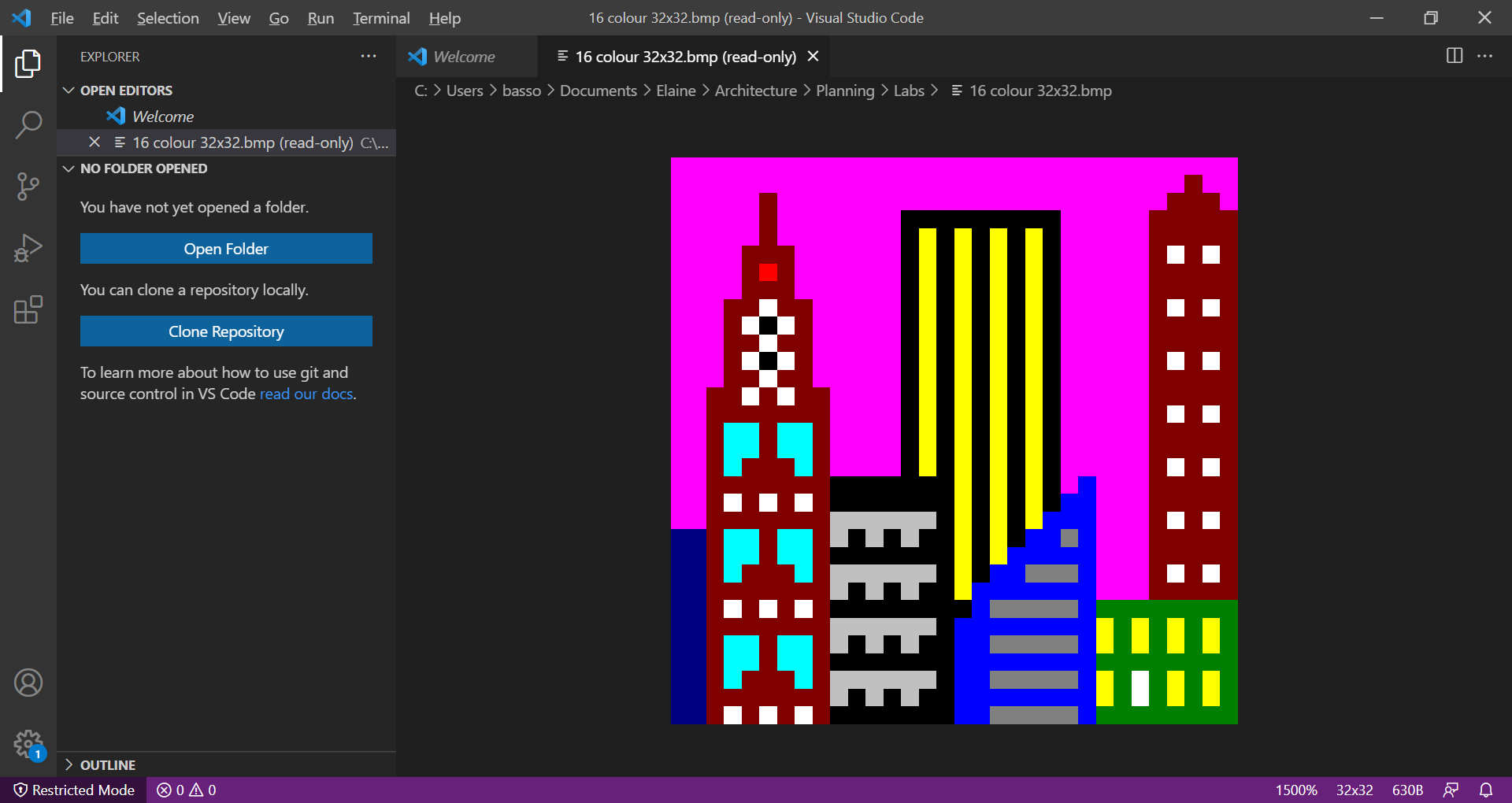
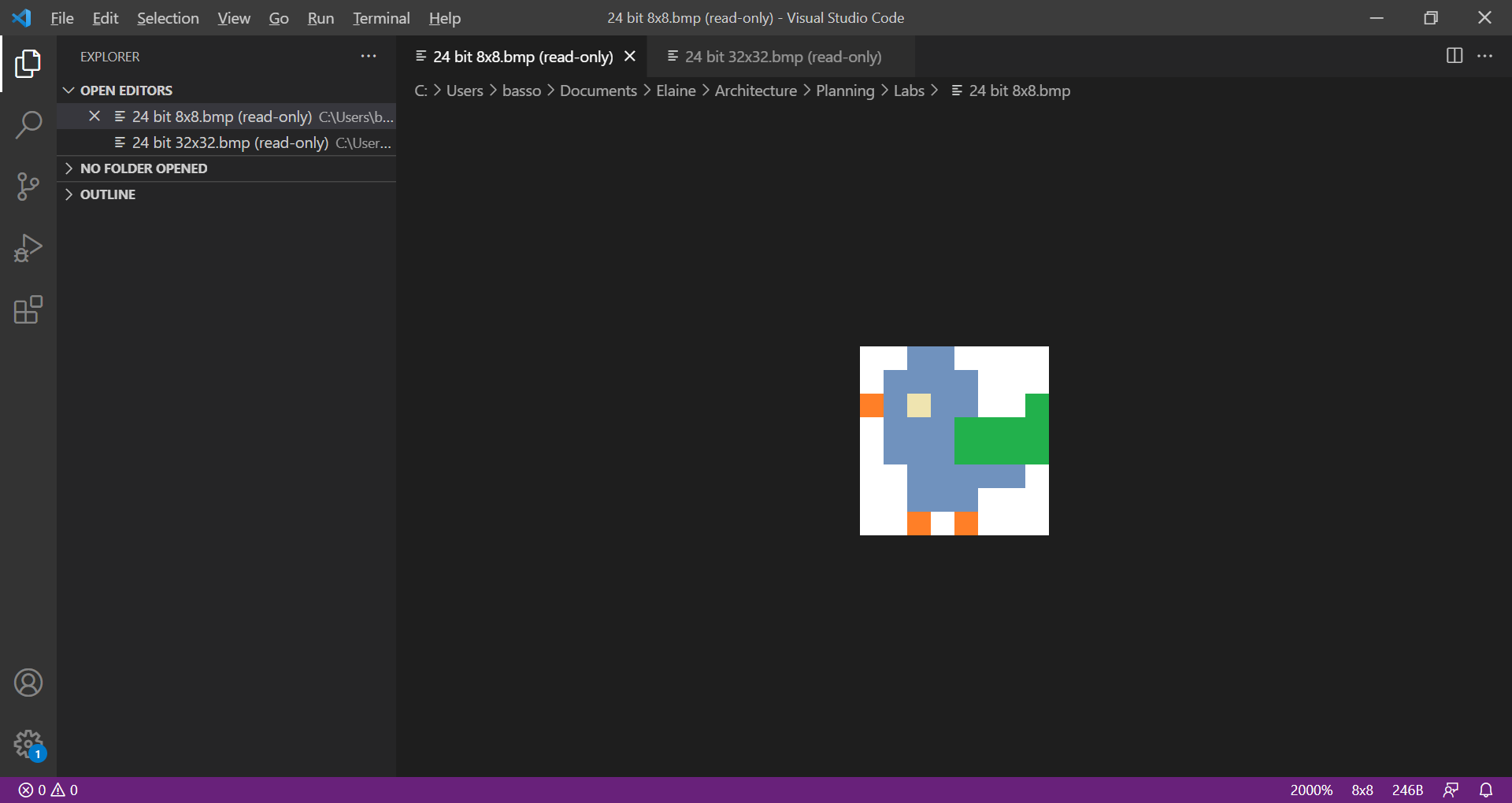
### Image representation using bitmap (bmp) files

This exercise looks at colour depth. Download the files labelled as for extension from Moodle.

In the lab, you looked at a bitmap image that used 24 bits for each pixel. That is each pixel had an RGB value where each of R, G and B take a value between decimal 0 and 255 which is equivalent to 00 to FF in hex. We saw that the pixel information in a bmp file starts at the bottom left of the image and is actually stored as BGR rather than the more usual RGB.

There are other ways that bitmap image data can be formatted, so we will look at these in more detail in these exercises.

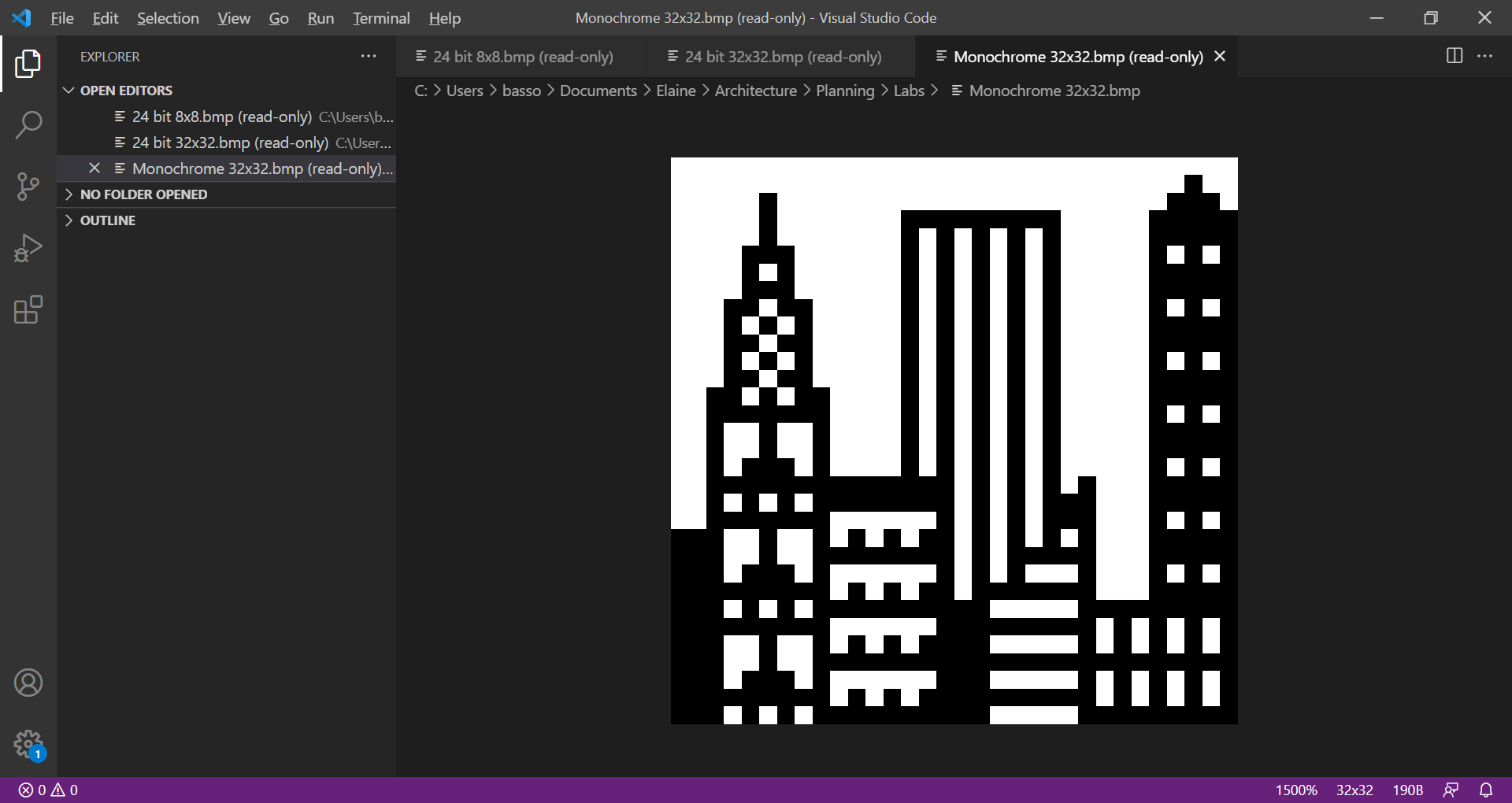
I have created the following images for these investigations:

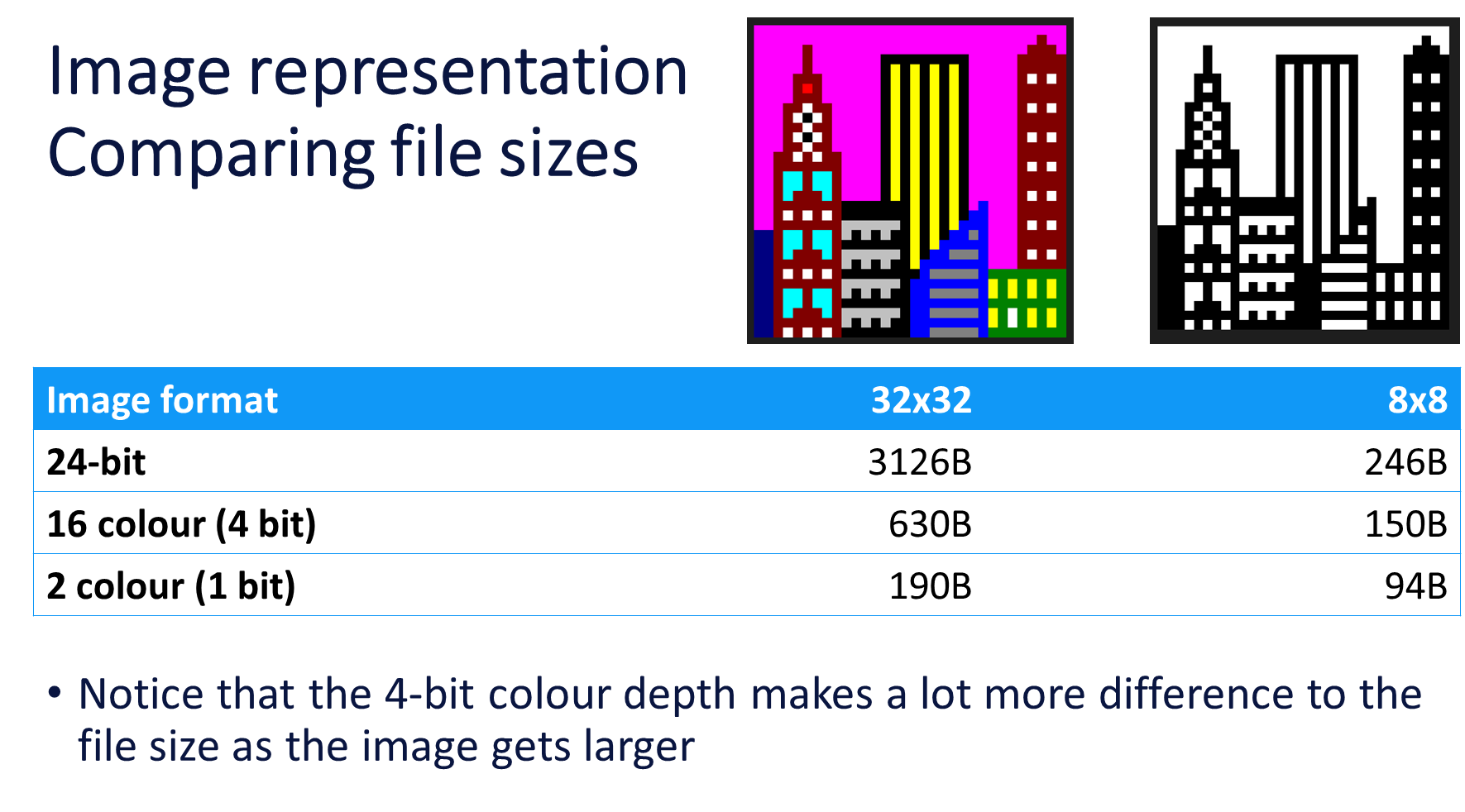
32x32 pixel 8x8 pixel

The files 8x8\_24\_bit.bmp and 32x32\_24\_bit.bmp are available in the folder and are encoded in the same way as you saw in the lab exercise above.

There are also black and white images (1 bit) as shown below and files that are stored in 16 colour format (4 bit). Download all the image files for the 8x8 and 32x32 images.



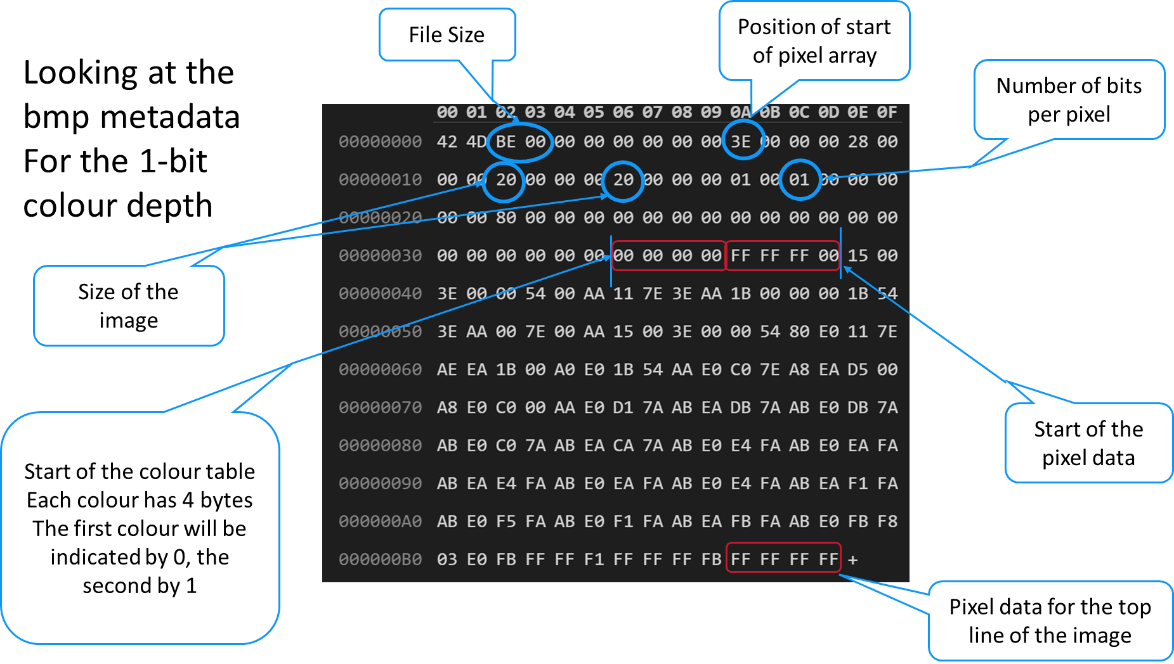
* 1. Look at the properties of the bitmap files in the different formats to see the file sizes. Compare them to each other. Open the files in VS Code so that you can see the images with the image preview. Which format would you prefer and why? Is this representative of real-life images?



* 1. Looking at the encoding for the 1-bit image. Open 32x32.bmp using the hex editor in VS Code. Note that there are two colours (black and white in this case) and each pixel can only be one of those two colours, that means we only need one bit for each pixel’s data.

When a bitmap has a restricted colour palette (in this case 2 colours), the metadata contains an indexed list of colours, or colour table. These colours are specified as 4 bytes (BGRx where x is a spare byte). Indexed means that the colour of a pixel can be looked up. In this case the pixel data can have a 0 or 1 for each pixel and it can look up the actual colour to use as the metadata will have a colour for 0 and one for 1.

* + 1. Look at the meta data and find out where the pixel data starts. Use exactly the same method as in the lecture/lab with the 24-bit encoding.



* + 1. The colour table specifying the two colours are the 8 bytes immediately before the start of the pixel data. They are currently 00 00 00 00 and FF FF FF 00 representing black and white to be used for colours 0 and 1 respectively.

Find those positions and change the two colours so that the picture is yellow and blue instead of black and white (or another two colours of your choice). Save your changes (Ctrl+S on Windows) and reopen the file as an image preview.

Changing the colours in the colour table

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

* + 1. Reopen the file with the hex editor again. Now we are going to change the image itself by changing the pixel data. Currently there is only one bit for each pixel (0 or 1) to indicate which colour to use but in the hex editor we see a hex digit (0 to F) for each four bits.

The aim is to extend the tops of the two tallest towers to reach the top of the image. Note that the black border in the figure above is not part of the image, so the top line of pixels in the image is all white. We know that the top line of the image is the last part of the pixel data. The image is 32 pixels wide, so there will be 32 bits representing the colours, so that is the last FF FF FF FF in the file is the pixel data for the top line of the image.

If we wrote that as binary, it would be 11111111111111111111111111111111 (32 1’s).

We need to work out which of those we want to change from 1 to 0 and then convert back to hex. Remember you can use a look up table when converting between binary and hex and with one hex digit for each group of 4 bits. You are expected to do some rough work for this.

The tower on the left will also need a change on the line underneath. Have a go at making those changes in the hex editor, save the file and open as an image preview.

Make any other changes you want to the image, plan your changes before manipulating the hex digits representing the pixel data.

Changing the top line

* We want to change the pixels at the following positions from 1 to 0
  + 6 pixels in from the left
  + 3 pixels in from the right
* In binary that gives a new top line of

1111 1011 1111 1111 1111 1111 1111 1011

* Now we need to convert it back to hex, we can look up that 1011 is B
* New pixel values for the top line FB FF FF FB

A screenshot of a computer

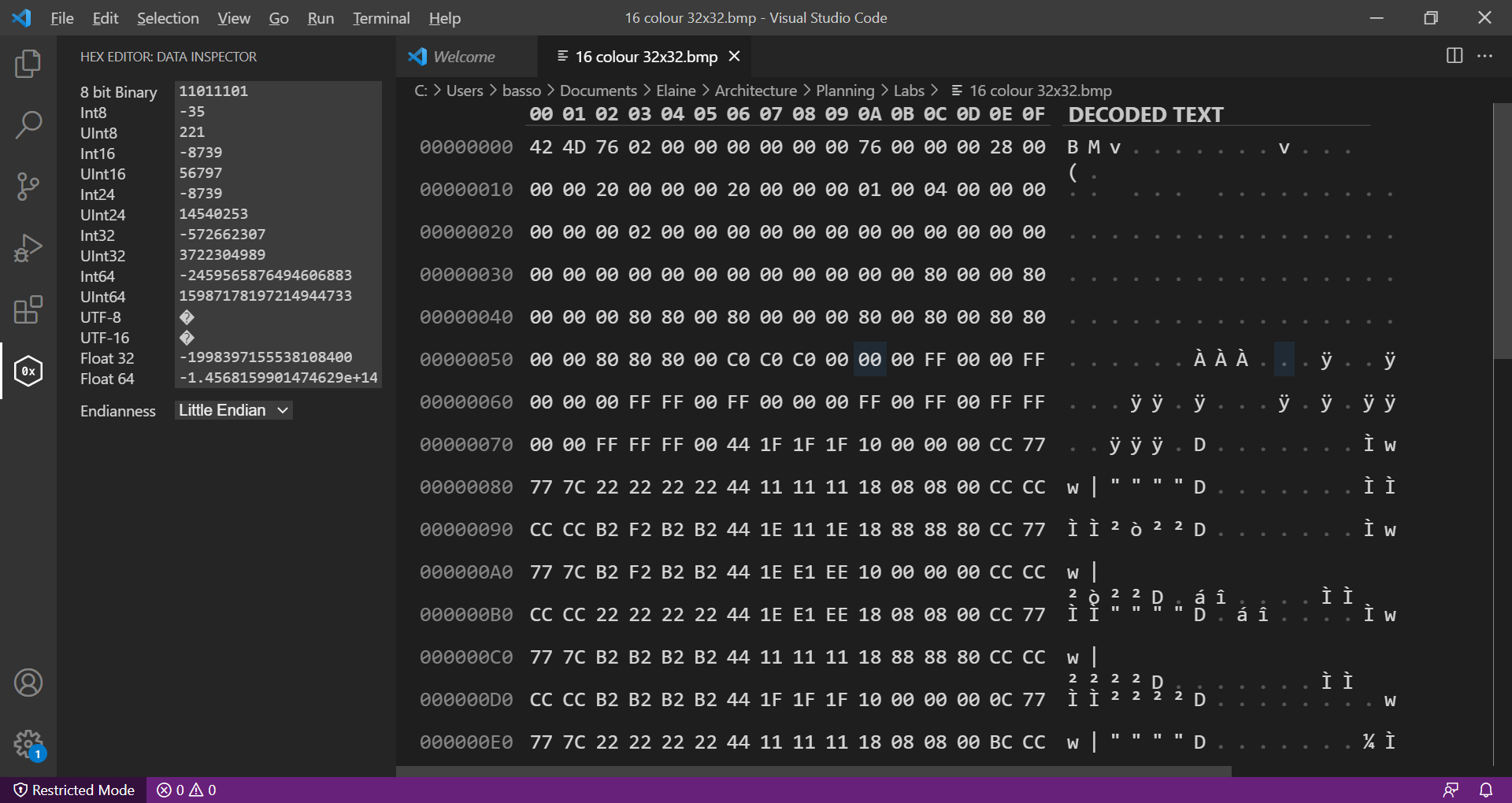
Description automatically generated

A screenshot of a computer

Description automatically generated

* 1. Looking at the 16 colour image. Visually, the image looks the same as the 24-bit one, but the file size is different. This image has a similar format to the 1-bit format described above, but now the colour index table has 16 colours. The pixel data has 4 bits for each pixel.

The image below shows some important parts of the bmp data with some of the specifications for colours in the colour table ringed in red. Note that only part of the pixel data is shown.



File Size

Position of start of pixel array

Number of bits per pixel

Size of the image

Start of the colour table

Start of the pixel data

In the hex editor, the file size shows as 76 02. To understand this, we need to introduce endianness. We have seen that the byte is a fundamental unit of data in computing. When we have information that requires multiple bytes, there is a fundamental decision that designers of processors have to make, what order to store the separate bytes in. That is the lowest order byte could come first in multiple bytes (little endian) or the highest order byte could come first (big endian). Most modern PCs and Macs are little endian, but other systems, like IBM zSeries are big endian.

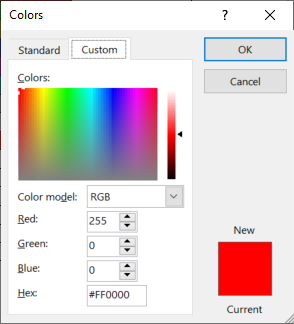
This means that this work is on a little endian system, so the actual size of the file in hex is 276. You should convert this to decimal and check against the file sizes you noted earlier.

* + 1. Investigate the colour table used in the image. What do the 16 different colours look like? Remember that each colour is specified using four bytes, but we are only interested in the first three that are in the order BGR.

You might want to create a table like that started below to record your findings:

|  |  |  |  |
| --- | --- | --- | --- |
| Index | BGR colour | RGB colour | Colour |
| 0 | 00 00 00 | 00 00 00 |  |
| 1 | 00 00 80 | 80 00 00 |  |
|  |  |  |  |
|  |  |  |  |

You can specify colours using RGB values in hex in Microsoft products. The hex value is preceeded by a # as you can see in the image below. The format with the # is commonly used for specifying colours, for example in CSS for applying style to websites.



* + 1. Each pixel has 4-bits in the pixel table to specify its colour, that is one hex digit. Make the following changes to the image by changing the relevant hex digits, referring to the table you created above:
    - Change the windows that are cyan to be bright green
    - Change the windows in the right-hand tower to be cyan instead of white
    - Change the windows that are dark grey to be dark yellow
    - Change the walls of the building on the bottom right (the one that has one white window and the rest yellow) to be dark cyan instead of dark green.

It might help to save the image after each change and check with the image preview.

|  |  |  |  |
| --- | --- | --- | --- |
| Index | BGR colour | RGB colour | Colour |
| 0 | 00 00 00 | 00 00 00 |  |
| 1 | 00 00 80 | 80 00 00 |  |
| 2 | 00 80 00 | 00 80 00 |  |
| 3 | 00 80 80 | 80 80 00 |  |
| 4 | 80 00 00 | 00 00 80 |  |
| 5 | 80 00 80 | 80 00 80 |  |
| 6 | 80 80 00 | 00 80 80 |  |
| 7 | 80 80 80 | 80 80 80 |  |
| 8 | C0 C0 C0 | C0 C0 C0 |  |
| 9 | 00 00 FF | FF 00 00 |  |
| A | 00 FF 00 | 00 FF 00 |  |
| B | 00 FF FF | FF FF 00 |  |
| C | FF 00 00 | 00 00 FF |  |
| D | FF 00 FF | FF 00 FF |  |
| E | FF FF 00 | 00 FF FF |  |
| F | FF FF FF | FF FF FF |  |

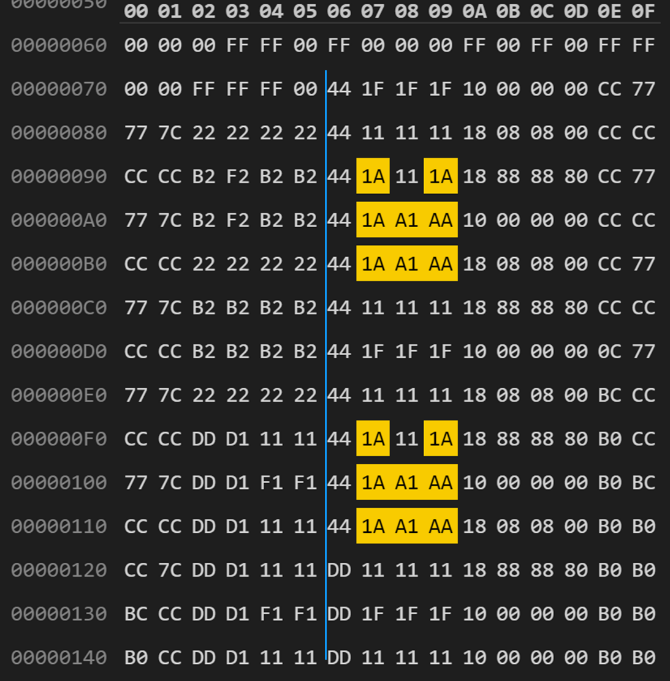
Table of colours from the original file

Changing the image

* Referring to the colour table and the pixel data
* Change the windows that are cyan to be bright green
* From colour table:
  + Cyan was E
  + Bright green was A
* BMP file format starts the pixel data at the bottom of the image
* The first cyan pixel is on the 3rd row up and the 4th pixel from the left
* Each row of pixels needs 32 hex digits – 16 bytes
* The hex editor shows 16 bytes across so each row of pixel data starts in the same column on the hex editor.



After changing E to A

 A screenshot of a computer

Description automatically generated

You can use a similar approach for the other changes.