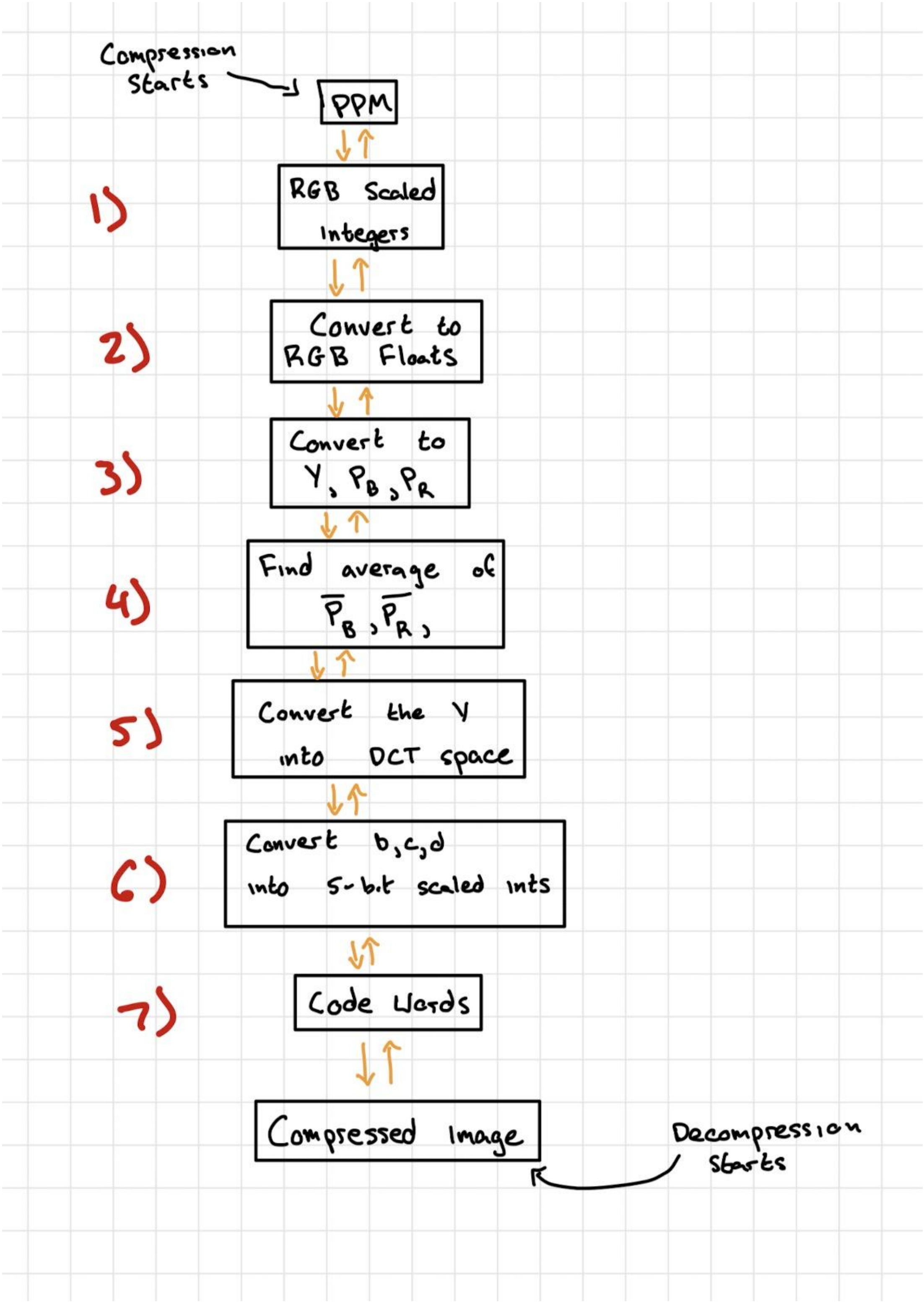
Stages for Arith:


Compression Starts
PPM
1)
RGB Scaled Integers
2)
Convert to RGB Floats
3)
Convert to Y, P_B, P_R
4)
Find average of $\overline{P}_B$, $\overline{P}_R$,
5)
Convert the Y into DCT space
6)
Convert b,c,d into 5-bit scaled ints
7)
Code Words
Compressed Image
Decompression Starts

```
Struct AvData
{
                A;
                B;
                C;
                D;
                $P_B$;
                $P_R$;
}
Struct VCS_data
{
        Y;
        $P_B$;
        $P_{R;}$
}
```

<u>Implementation</u>:

Each step of compression and decompression will be done together (1 of compression, 1 of decompression) so that we can test each step along the way.

Each pair of steps will be contained in its own file

<u>Compression:</u>

1. **Make UArray2b with pixel data in RGB color space**
    a. Inputs: Pnm File
    b. Outputs: UArray2b with pixel data from PNM data set.
    c. No information is lost
2. **Convert the RGB to Video Color Space**
    a. Inputs: UArray2b with pixel data in RGB colorspace
    b. Output: UArray2b with pixel data in video color space
    c. Details: Updates the information stored in each element of the UArray2B
        i. y = 0.299 * r + 0.587 * g + 0.114 * b;
        ii. pb = -0.168736 * r - 0.331264 * g + 0.5 * b;
        iii. pr = 0.5 * r - 0.418688 * g - 0.081312 * b;
    d. No information is lost because these values can be converted from one to another using this formula

3. **Calculate the average P_B and P_R**
    a. Input: UArray2b with VCS data for each pixel
    b. Output: UArray2 with VCS_data averages for each block
    c. Details:
        i. Create a UArray2 with size (width / block size) and (height / blocksize)
        ii. Each element will be a VCS_data struct;
        iii. Blockmajor mapping over input
            1. Test: Print function to test these averages
    d. Information is lost because we are taking the average of the data over each block rather than each pixel in each block.
4. **Convert the P_B and P_R elements to four-bit values**
    a. Input: UArray2 from last step
    b. Output: UArray2 with changes to member data
    c. Details:
        i. Map over UArray2
        ii. Get each struct inside
        iii. Convert each P_B and P_R to four-bit values : unsigned Arith40_index_of_chroma(float x);
        iv. Store these back in P_B and P_R
    d. Information is lost because we are storing information that is contained in more bit into information that is stored in less bits using quantized buckets

5. **DCT**
   a. Input: UArray2B from step 2, and UArray2 from step 4
   b. Output: UArray2 from step 4 with data members set.
   c. Details: where Y1-4 are each pixel 1-4 block major wise in image.
      i. a = (Y4 + Y3 + Y2 + Y1)/4.0
      ii. b = (Y4 + Y3 − Y2 − Y1)/4.0
      iii. c = (Y4 − Y3 + Y2 − Y1)/4.0
      iv. d = (Y4 − Y3 − Y2 + Y1)/4.0
   d. No information is lost in this step because this is entirely reversible, and no data is left out or does not have a mapping.
6. **Convert b, c ,d to scale between -0.3 and 0.3**
   a. Input: 9 bit value
   b. Output: 5 bit value
   c. Details:
      i. Function takes in 9 bit value
      ii. Divide 9 bit value by the scale
      iii. Cast to 5 bit value
      iv. update b, c, d to 5 bit value
   d. Information is lost because we are casting down to less bits to represent the same amount of data.
7. **Make Word**
   a. Input: Struct Av_data
   b. Gets each value and casts it to the correct size:
      i. A -> 9 bits
      ii. B -> 5 bits
      iii. C -> 5 bits
      iv. D -> 5 bits
      v. P_B 4 bits
      vi. P_R 4 bits0
   c. Compose words by putting these bits together.
8. **Make all the words output in sequence AND all other file IO stuff**

<u>Decompression:</u>

The inputs and outputs for the decompression phase are the reverse of the compression phase. The information loss will only occur in compression too.

1. **Remake UArray2b structure**
   a. Read over full sequence, divide length of sequence by 32 to get number of words
   b. Create UArray2b with 2 block size and width and height of sqrt(num_words)
   c. Create apply function that takes in the sequence and runs Step 2
      i. Sets each AvData elem in UArray2b to data read from Step 2
2. **Get data from Word:**

a. Takes in 32 bit word
b. Creates struct
c. Create Function that gets a specified number of bits from the word
d. First 9 bits of word are stored in A
e. Next 5 bits are stored in B
f. Next 6 bits are stored in C
g. Next 5 Bits are stored in D
h. PB and PR are the subsequent 4 bit sequences
    i. use float Arith40_chroma_of_index(unsigned n);
i. Returns S2

3. **Reverse DCT to get Y1, Y2, Y3, Y4**
    a. Y1 = a − b − c + d
    b. Y2 = a − b + c − d
    c. Y3 = a + b − c − d
    d. Y4 = a + b + c + d
4. **Convert the PB, PR, a, b, c, d values from ints to floats**
    a. Go over each AvData member and convert to float
5. **Convert the Video Color Space to RGB**
    a. Transform the pixel from component-video color to RGB color
    b. Quantize the RGB values to integers in the range 0 to 255
    c. Put the RGB values into pixmap->pixels using apply function
6. **Use PPMwrite to output the ppm file**

<u>Testing:</u>

1. For the functions to go from RGB to Video Color Space and vice versa
    a. Use the functions to test the output of each other
        i. Given a set of RGB values, the same values should be returned after getting converted to video color space and converted back to RGB
2. For the functions to convert the PB and PR values to the average PB and PR and vice versa
    a. Make use of the given conversion functions provided to convert to and from the average values
        i. The PB and PR values given should be returned after running through both functions.
3. For the functions to transform pixel space to DCT space and vice versa
    a. Make use of the given equations to create a function that takes the current pixel space and transforms it into DCT space
    b. The inverse function will be created using the DCT space to pixel space equations
        i. The pixel values inputted should be returned after running through both functions

4. For the functions to convert the b,c, and d values to 5-bit scaled signed ints and vice versa
    i. Use the two functions to test the output of the other. The output for one of the functions should be the input for the other.
5. For the packing functions
    i. Make use of the unpacking and packing functions to test the input and output of each other.