Basic concept of image compression technique: JPEG

Encoding and Decoding Mechanism

What is Data Compression?

- Data compression requires the identification and extraction of source redundancy. In other words, data compression seeks to reduce the number of bits used to store or transmit information.

WHY IMAGE COMPRESSION?

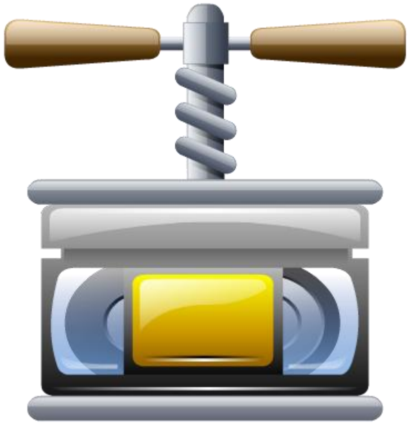
- Reducing the amount of data
- Reducing transmission time

Types of Image Compression

Lossless Image Compression

- **No image data is lost** in this type of compression, it can be recovered again by uncompressing the file using the formula.

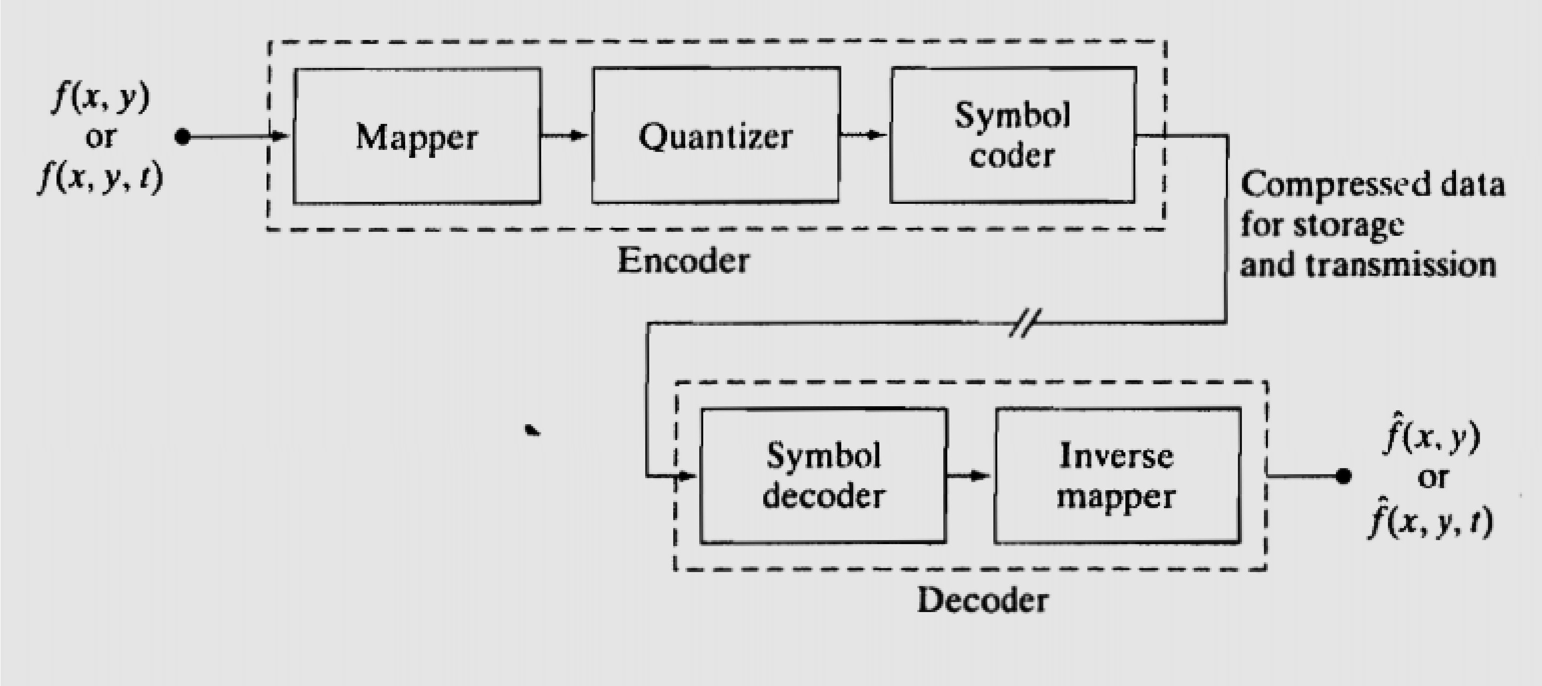
- Lossless compression normally produces **better quality** images but they will be bigger image sizes than those using lossy image compression.



Lossy Image Compression

- **Lossy image compression** is the type of compression that results in you losing some level of quality in the image file.
- This image data is **permanently lost**, the file will never be as good as the original again.
- Images that are compressed using this technique are much **smaller** in file size.

Block Diagram of Image Compression



THE JPEG DEFINED

JPEG stands for Joint Photographic Experts Group — an international image compression standard invented in 1992.

JPEG can compress images of any size and any resolution with acceptable consideration of **quality-compression trade-off**.

JPEG can be either **lossy** or **lossless**, although **lossless JPEG** is not in extensive use.

Lossy JPEG, alternately known as **Baseline JPEG**, is extremely popular in computer imaging.

JPEG OVERVIEW

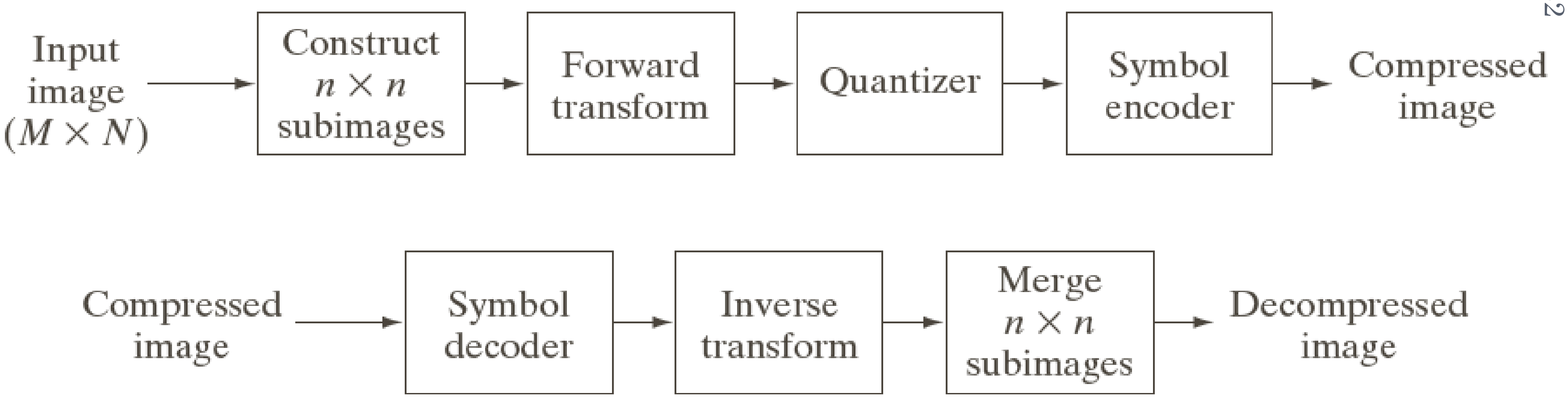
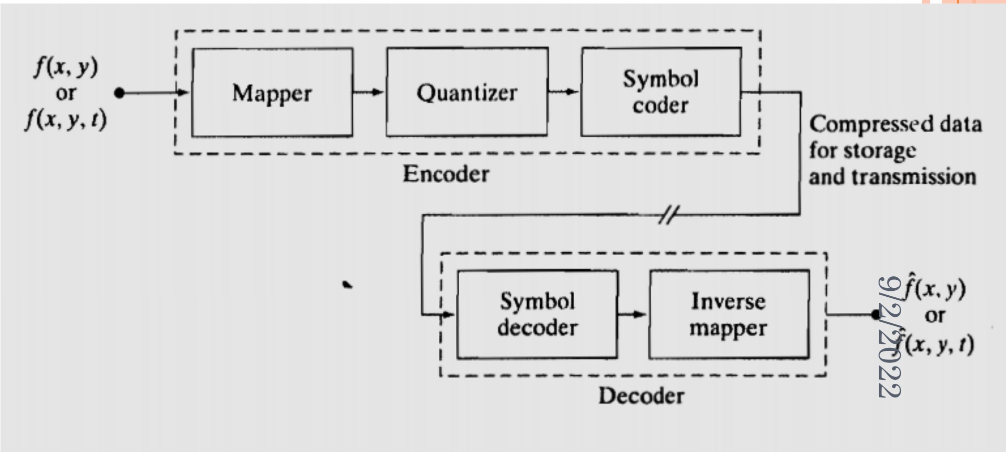


Fig 1: JPEG Encoder and Decoder

IMAGE COMPRESSION

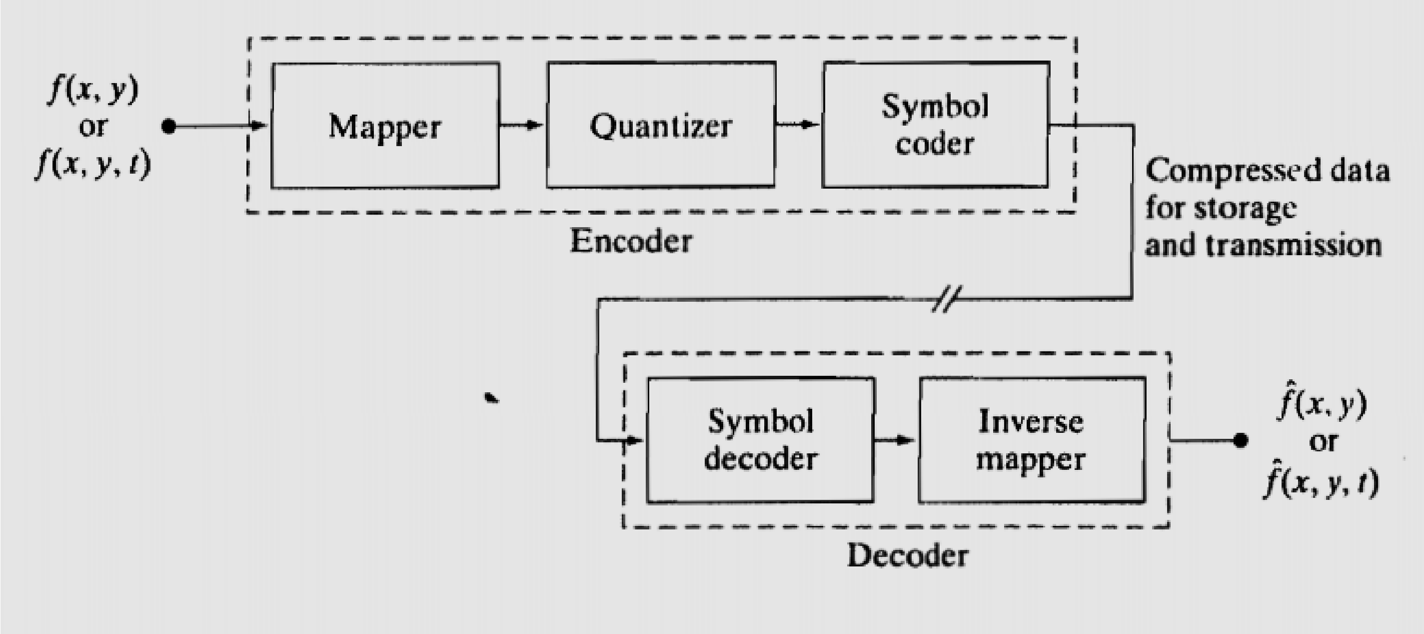


- The compression algorithm can be broken into the following stages:

Mapping: Involves mapping the original image data into another mathematical space where it is easier to compress the data.

Quantization: Involves taking potentially continuous data from the mapping stage and putting it in discrete form

Coding: Involves mapping the discrete data from the quantizer onto a code in an optimal manner



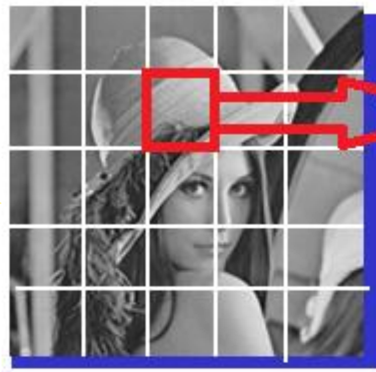
The decompression can be broken down into following stages:

- ***Decoding***: Takes the compressed file and reverses the original coding by mapping the codes to the original quantized values
- ***Inverse mapping***: Involves reversing the original mapping process

JPEG-Encoder

Input Image → Construct 8*8 sub-image → Forward Transform → Quantization → Symbol Coder → Output Image

Original Image



8*8 sub-image

One block

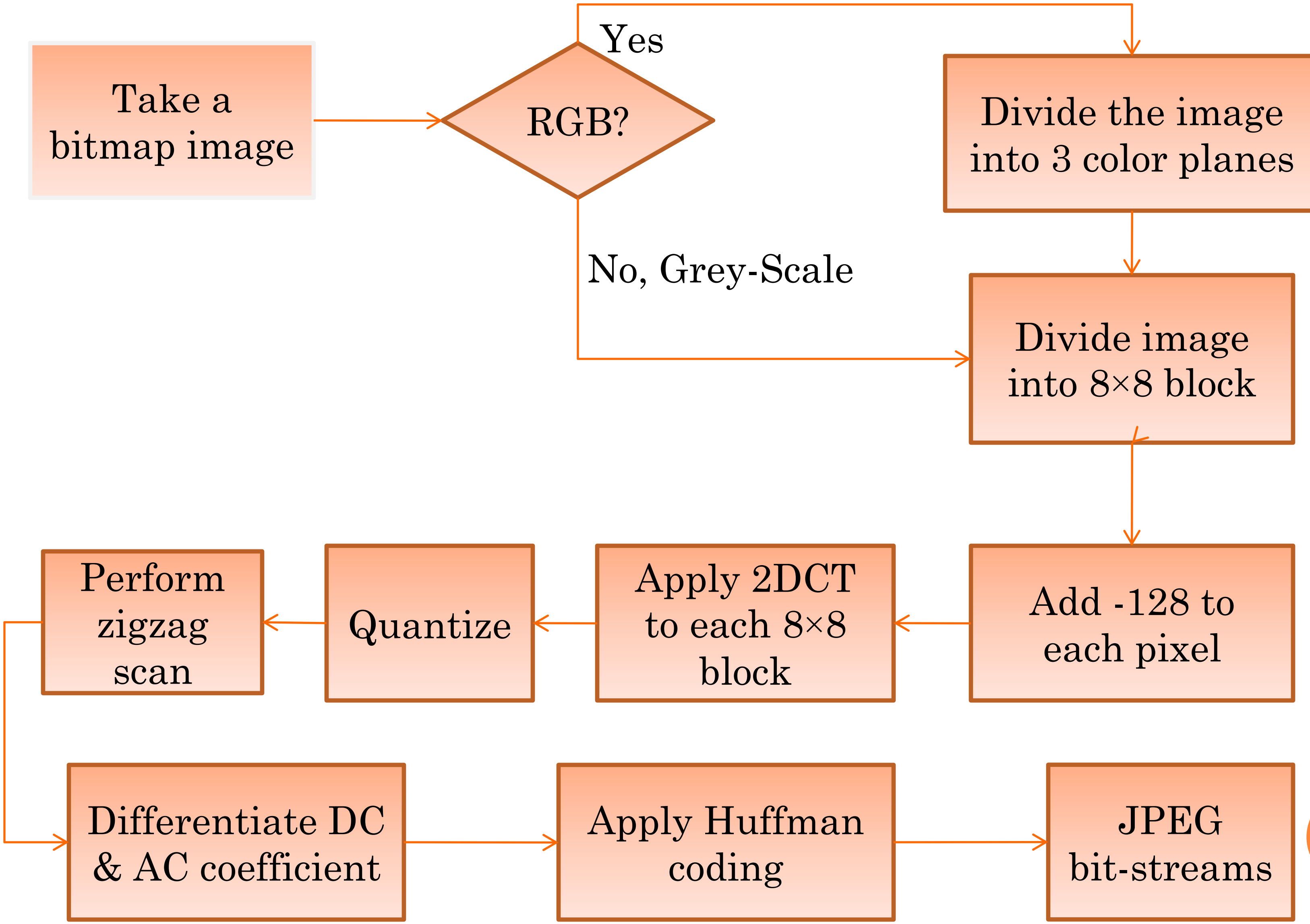
214	224	199	190	222	231	239	236
215	198	179	218	219	223	229	236
202	166	205	215	218	208	144	183
173	177	211	212	200	152	136	196
155	197	205	200	143	133	178	221
158	197	198	153	118	159	202	211
169	192	167	110	139	190	196	199
169	180	131	105	172	193	190	191

Shift operation

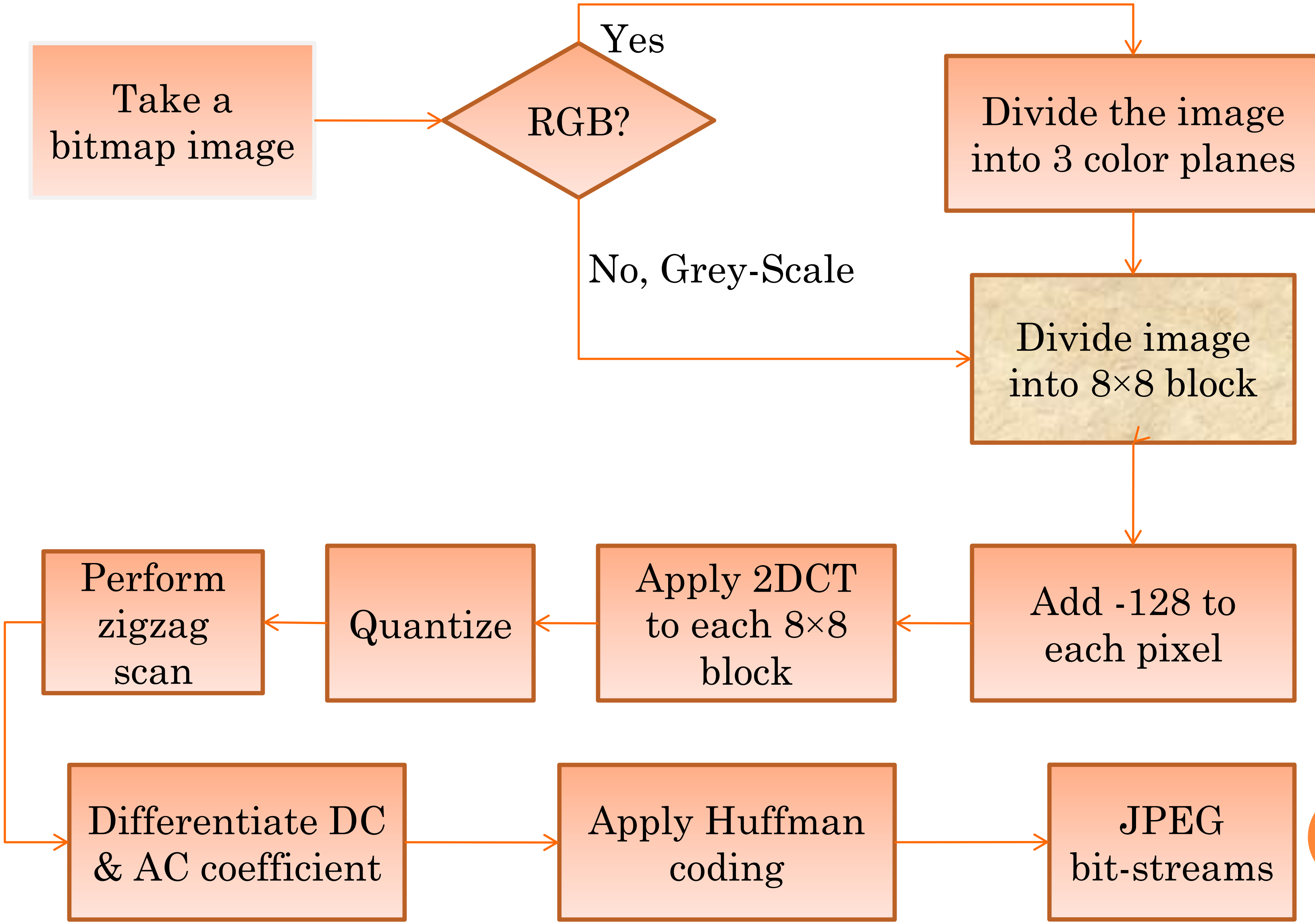
-128

86	96	71	62	94	103	111	108
87	70	51	90	91	95	101	108
74	38	77	87	90	80	16	55
45	49	83	84	72	24	8	68
27	69	77	72	15	5	50	93
30	69	70	25	−10	31	74	83
41	64	39	−18	11	62	68	71
41	52	3	−23	44	65	62	63

The baseline JPEG Algorithmic Flow-Chart



Specified Example of JPEG Compression

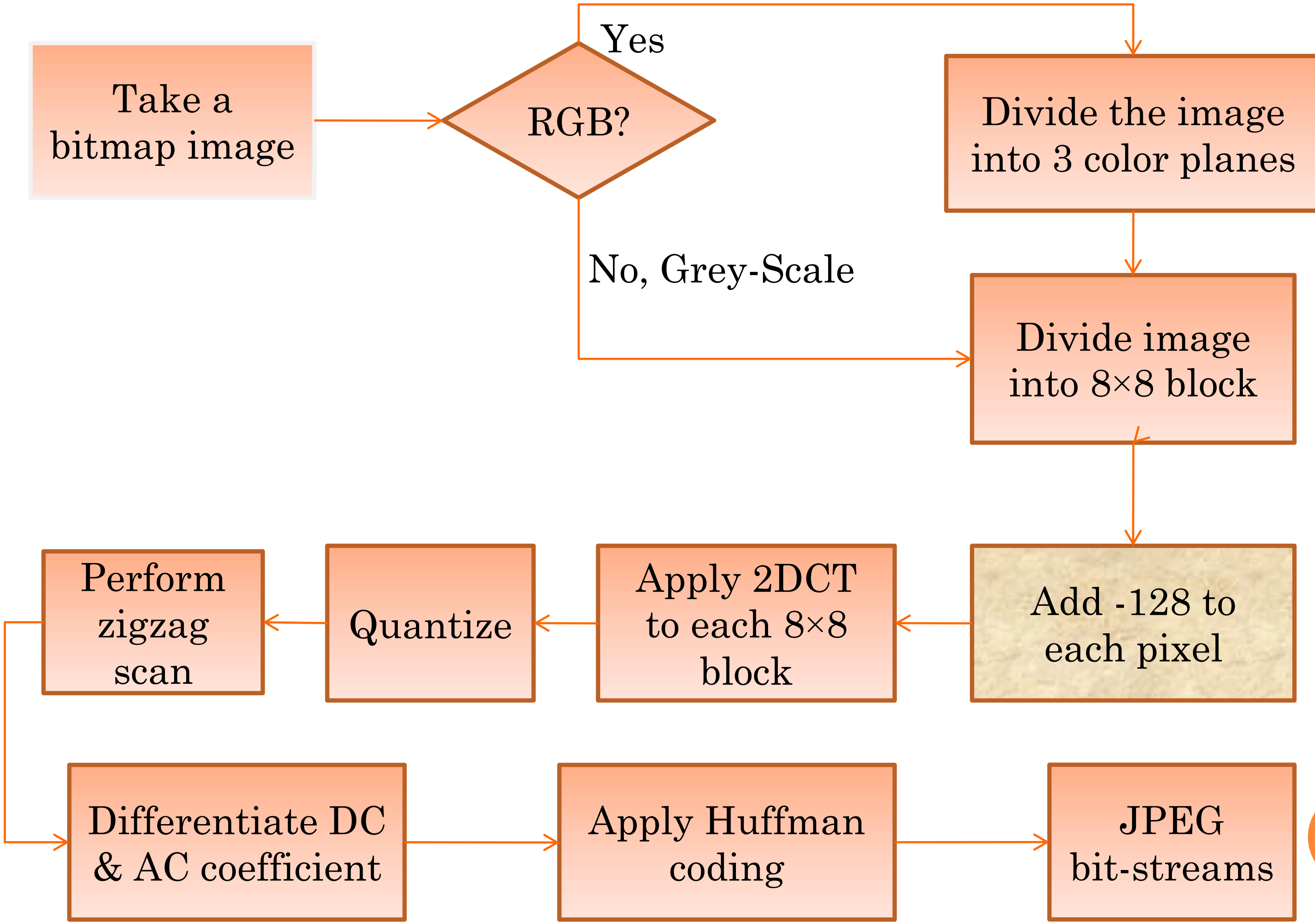


Example Continued…

Let an 8×8 block of a bitmap image while performing JPEG algorithm on it.

$$\begin{bmatrix} 52 & 55 & 61 & 66 & 70 & 61 & 64 & 73 \\ 63 & 59 & 55 & 90 & 109 & 85 & 69 & 72 \\ 62 & 59 & 68 & 113 & 144 & 104 & 66 & 73 \\ 63 & 58 & 71 & 122 & 154 & 106 & 70 & 69 \\ 67 & 61 & 68 & 104 & 126 & 88 & 68 & 70 \\ 79 & 65 & 60 & 70 & 77 & 68 & 58 & 75 \\ 85 & 71 & 64 & 59 & 55 & 61 & 65 & 83 \\ 87 & 79 & 69 & 68 & 65 & 76 & 78 & 94 \end{bmatrix}.$$

Algorithmic Step



EXAMPLE CONTINUED…

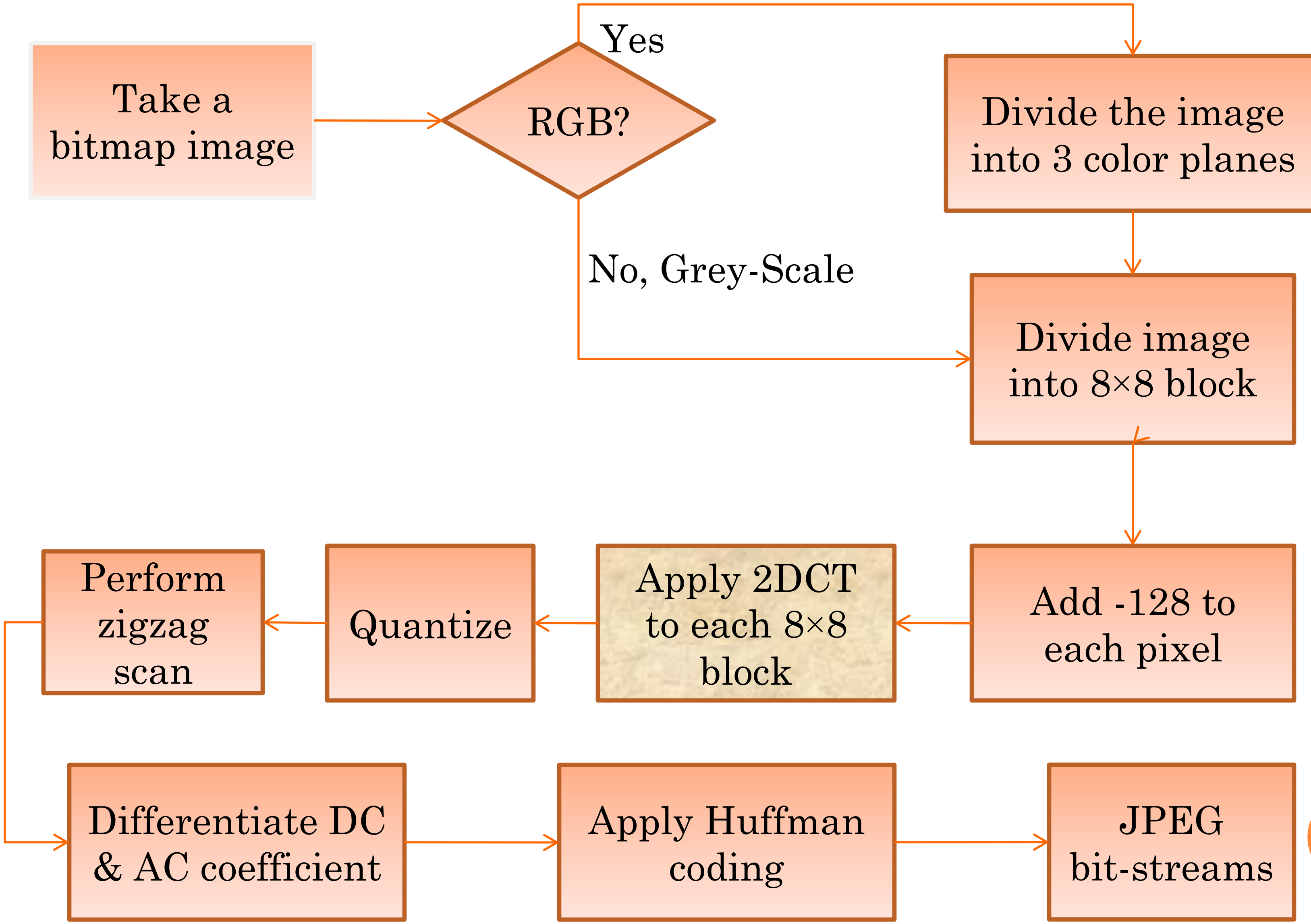
We add -128 to each pixel in this step.

This step reduces the Dynamic Range (reduces the volume of loud sounds or amplifies quiet sound) requirements in the next DCT processing stage.

$$\begin{bmatrix} 52 & 55 & 61 & 66 & 70 & 61 & 64 & 73 \\ 63 & 59 & 55 & 90 & 109 & 85 & 69 & 72 \\ 62 & 59 & 68 & 113 & 144 & 104 & 66 & 73 \\ 63 & 58 & 71 & 122 & 154 & 106 & 70 & 69 \\ 67 & 61 & 68 & 104 & 126 & 88 & 68 & 70 \\ 79 & 65 & 60 & 70 & 77 & 68 & 58 & 75 \\ 85 & 71 & 64 & 59 & 55 & 61 & 65 & 83 \\ 87 & 79 & 69 & 68 & 65 & 76 & 78 & 94 \end{bmatrix}.$$

$$\begin{bmatrix} -76 & -73 & -67 & -62 & -58 & -67 & -64 & -55 \\ -65 & -69 & -73 & -38 & -19 & -43 & -59 & -56 \\ -66 & -69 & -60 & -15 & 16 & -24 & -62 & -55 \\ -65 & -70 & -57 & -6 & 26 & -22 & -58 & -59 \\ -61 & -67 & -60 & -24 & -2 & -40 & -60 & -58 \\ -49 & -63 & -68 & -58 & -51 & -60 & -70 & -53 \\ -43 & -57 & -64 & -69 & -73 & -67 & -63 & -45 \\ -41 & -49 & -59 & -60 & -63 & -52 & -50 & -34 \end{bmatrix}$$

Algorithmic Step



9/2/2022

Example Continued…

Discrete Cosine Transform should now be applied to each block as follows-

$$S_{vu} = \frac{1}{4} C_u C_v \sum_{x=0}^{7} \sum_{y=0}^{7} S_{yx} \cos \frac{(2x+1)u\pi}{16} \cos \frac{(2y+1)}{v\pi}$$

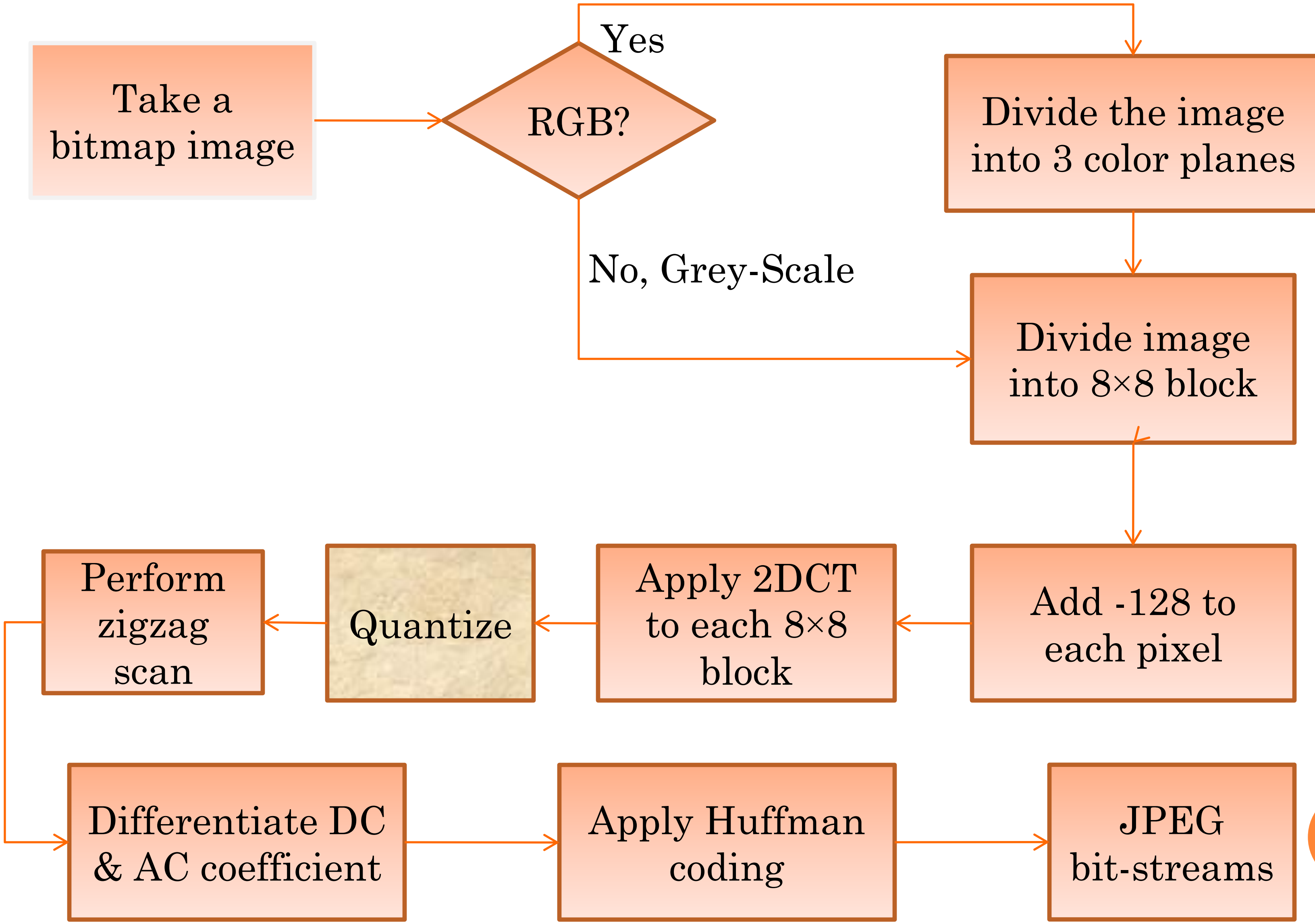
where $C_u, C_v = 1/\sqrt{2}$ for $u, v = 0$; otherwise $C_u, C_v = 1$ and $0 \leq u, v \leq 7$

$$\begin{bmatrix} -76 & -73 & -67 & -62 & -58 & -67 & -64 & -55 \\ -65 & -69 & -73 & -38 & -19 & -43 & -59 & -56 \\ -66 & -69 & -60 & -15 & 16 & -24 & -62 & -55 \\ -65 & -70 & -57 & -6 & 26 & -22 & -58 & -59 \\ -61 & -67 & -60 & -24 & -2 & -40 & -60 & -58 \\ -49 & -63 & -68 & -58 & -51 & -60 & -70 & -53 \\ -43 & -57 & -64 & -69 & -73 & -67 & -63 & -45 \\ -41 & -49 & -59 & -60 & -63 & -52 & -50 & -34 \end{bmatrix} \quad \begin{bmatrix} -415.38 & -30.19 & -61.20 & 27.24 & 56.13 & -20.10 & -2.39 & 0.46 \\ 4.47 & -21.86 & -60.76 & 10.25 & 13.15 & -7.09 & -8.54 & 4.88 \\ -46.83 & 7.37 & 77.13 & -24.56 & -28.91 & 9.93 & 5.42 & -5.65 \\ -48.53 & 12.07 & 34.10 & -14.76 & -10.24 & 6.30 & 1.83 & 1.95 \\ 12.12 & -6.55 & -13.20 & -3.95 & -1.88 & 1.75 & -2.79 & 3.14 \\ -7.73 & 2.91 & 2.38 & -5.94 & -2.38 & 0.94 & 4.30 & 1.85 \\ -1.03 & 0.18 & 0.42 & -2.42 & -0.88 & -3.02 & 4.12 & -0.66 \\ -0.17 & 0.14 & -1.07 & -4.19 & -1.17 & -0.10 & 0.50 & 1.68 \end{bmatrix}$$

Discrete Cosine Transform

- Why DCT is more appropriate for image compression than DFT?
  - The DCT has the **ability to pack most of the information in fewest coefficient**
  - For image compression, the DCT **can reduce the blocking effect** than the DFT

ALGORITHMIC STEP



EXAMPLE CONTINUED…

The JPEG defines **a quantization matrix for quantizing each block. The quality of a compressed image** can be controlled by the quantization matrix.

$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix} \times \text{Quality; default quality=1.}$$

Example Continued…

JPEG quantization is defined as **simply element wise dividing the obtained matrix by quantization matrix.**

$$
\begin{bmatrix}
-415.38 & -30.19 & -61.20 & 27.24 & 56.13 & -20.10 & -2.39 & 0.46 \\
4.47 & -21.86 & -60.76 & 10.25 & 13.15 & -7.09 & -8.54 & 4.88 \\
-46.83 & 7.37 & 77.13 & -24.56 & -28.91 & 9.93 & 5.42 & -5.65 \\
-48.53 & 12.07 & 34.10 & -14.76 & -10.24 & 6.30 & 1.83 & 1.95 \\
12.12 & -6.55 & -13.20 & -3.95 & -1.88 & 1.75 & -2.79 & 3.14 \\
-7.73 & 2.91 & 2.38 & -5.94 & -2.38 & 0.94 & 4.30 & 1.85 \\
-1.03 & 0.18 & 0.42 & -2.42 & -0.88 & -3.02 & 4.12 & -0.66 \\
-0.17 & 0.14 & -1.07 & -4.19 & -1.17 & -0.10 & 0.50 & 1.68
\end{bmatrix}
$$

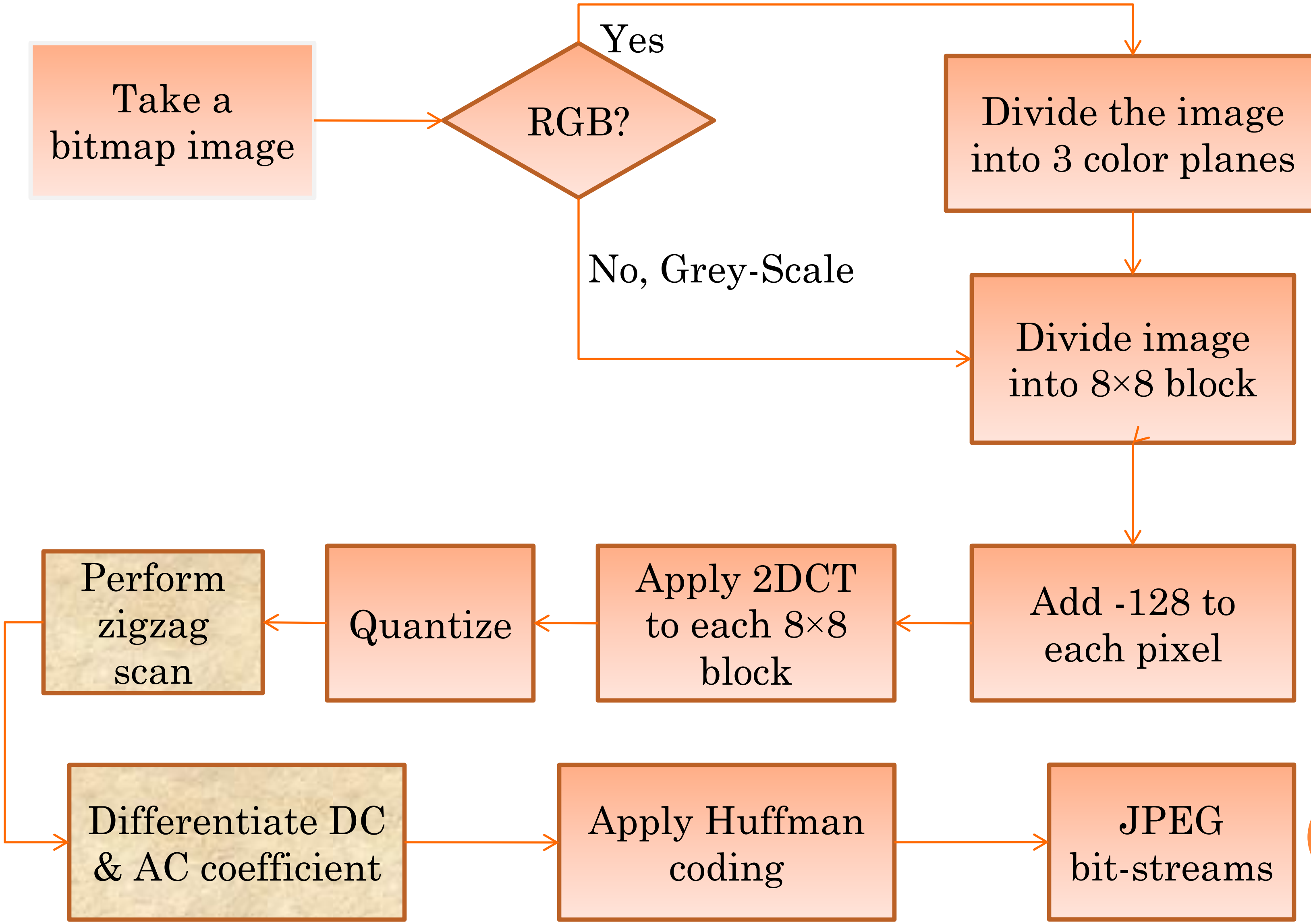
$$
B = \begin{bmatrix}
-26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\
0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\
-3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\
-3 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}.
$$

$$
Q = \begin{bmatrix}
16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\
12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\
14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\
14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\
18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\
24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\
49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\
72 & 92 & 95 & 98 & 112 & 100 & 103 & 99
\end{bmatrix}.
$$

$$
\mathrm{round}\left(\frac{-415.38}{16}\right) = \mathrm{round}(-25.96) = -26
$$

9/2/2022

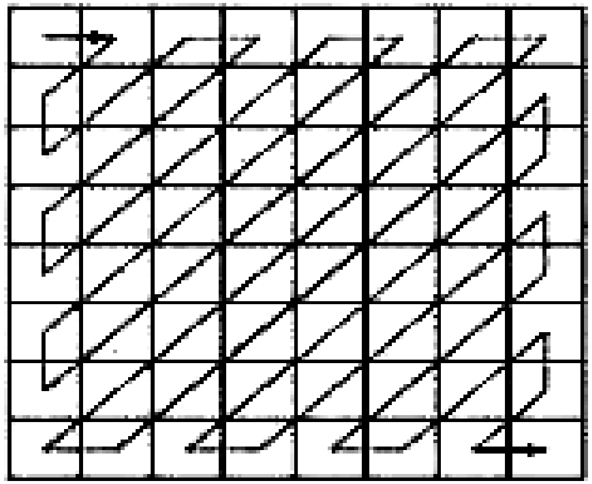
ALGORITHMIC STEP



EXAMPLE CONTINUED…

In this step, **zigzag ordering of the quantized block should be performed.**

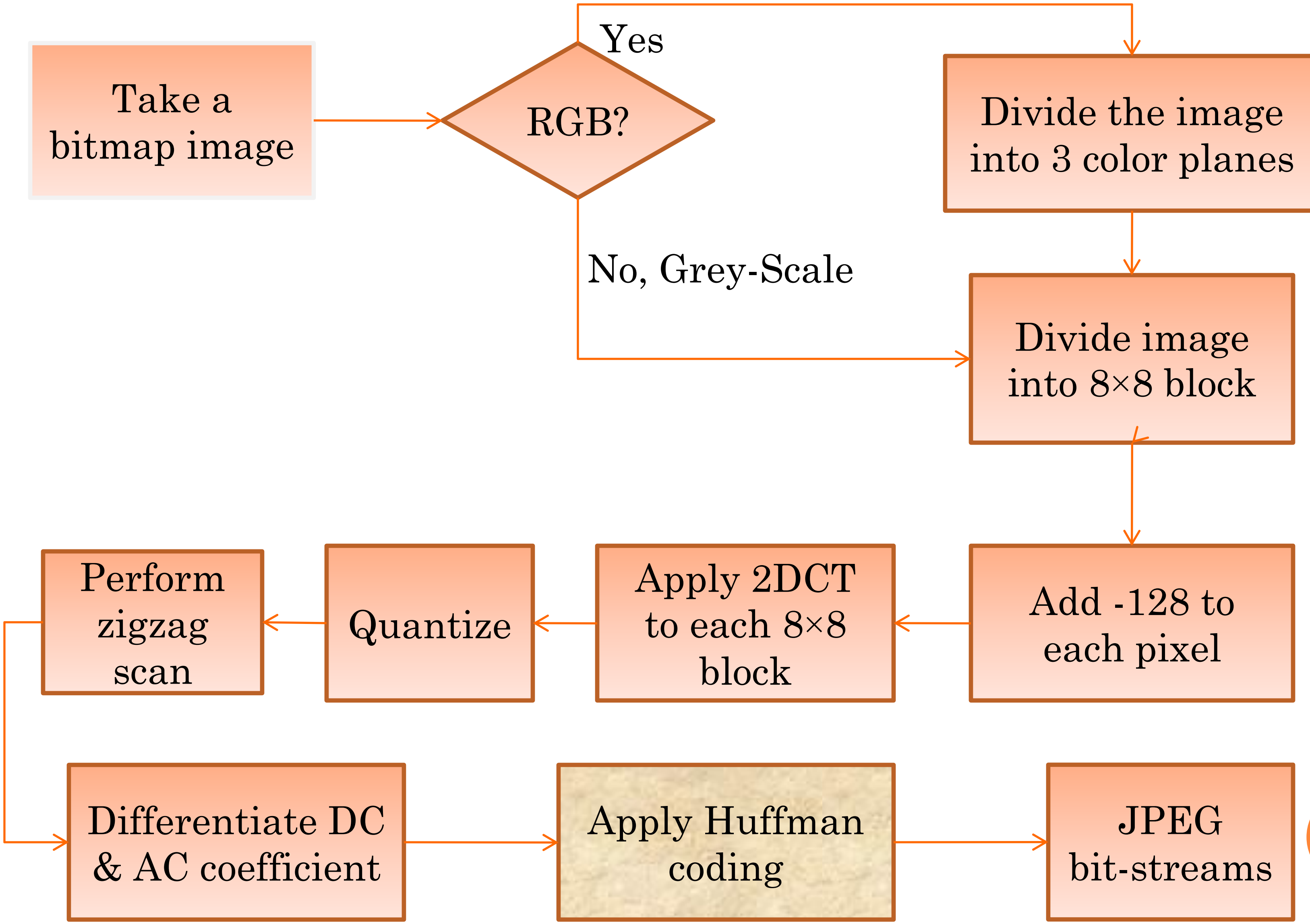
$$B = \begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -3 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$



-26,-3,0,-3,-2,-6,2,-4,1,-3,1,1,5,1,2,-1,1,-1,2,0,0,0,0,0,-1,-1,EOB

→ **DC Coefficient**
→ **AC Coefficient**
→ **End of Block**

ALGORITHMIC STEP



9/2/2022

EXAMPLE CONTINUED…

Applying Huffman Coding, we get a bit stream from the sequence of DC and AC coefficients as follows-

-26,-3,0,-3,-2,-6,2,-4,1,-3,1,1,5,1,2,-1,1,-1,2,0,0,0,0,0,-1,-1,EOB

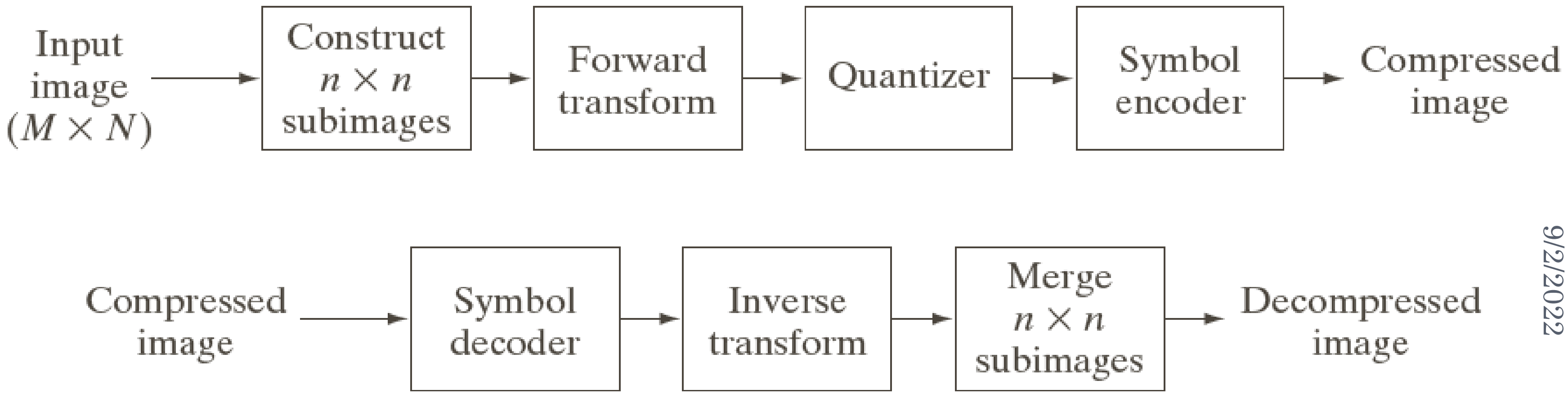
1010110 0100 11100100 0100 0101 100001 0110 100011 001 100011 001 001 100101 11100110 110110 0110 11110100 000 1010.

Termination

The technique shown in the previous example block should be continued unless all 8×8 blocks of the image are processed.

Thus a bitmap image is completely JPEG encoded.

When we view a JPEG image in a display device, we actually see the decoded JPEG.



✓ Huffman Coding

- The Huffman code, developed by D. Huffman in 1952, is a *minimum length code*
- This means that given the statistical distribution of the gray levels (the histogram), **the Huffman algorithm will generate a code that is as close as possible to the *minimum bound, the entropy***

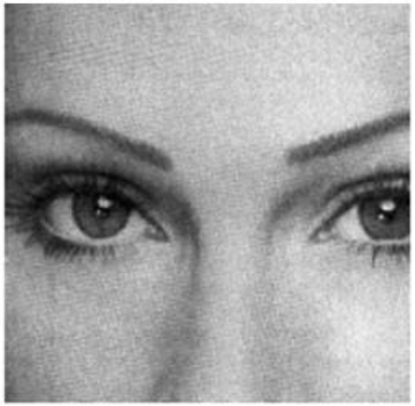
Original source		Source reduction			
Symbol	Probability	1	2	3	4
a_2	0.4	0.4	0.4	0.4	0.6
a_6	0.3	0.3	0.3	0.3	0.4
a_1	0.1	0.1	0.2	0.3	
a_4	0.1	0.1	0.1		
a_3	0.06	0.1			
a_5	0.04				

Original source			Source reduction							
Symbol	Probability	Code	1		2		3		4	
a_2	0.4	1	0.4	1	0.4	1	0.4	1	0.6	0
a_6	0.3	00	0.3	00	0.3	00	0.3	00	0.4	1
a_1	0.1	011	0.1	011	0.2	010	0.3	01		
a_4	0.1	0100	0.1	0100	0.1	011				
a_3	0.06	01010	0.1	0101						
a_5	0.04	01011								

The average code length is

$$
\begin{aligned}
L_{avg} &= (0.4)(1) + (0.3)(2) + (0.1)(3) + (0.1)(4) + (0.06)(5) + (0.04)(5) \\
&= 2.2 \text{ bits/pixel}
\end{aligned}
$$

Decoding



Original Image

Encoding by DCT

231	224	224	217	217	203	189	196
210	217	203	189	203	224	217	224
196	217	210	224	203	203	196	189
210	203	196	203	182	203	182	189
203	224	203	217	196	175	154	140
182	189	168	161	154	126	119	112
175	154	126	105	140	105	119	84
154	98	105	98	105	63	112	84

Inverse DCT

42	28	35	28	42	49	35	42
49	49	35	28	35	35	35	42
42	21	21	28	42	35	42	28
21	35	35	42	42	28	28	14
56	70	77	84	91	28	28	21
70	126	133	147	161	91	35	14
126	203	189	182	175	175	35	21
49	189	245	210	182	84	21	35

Decoding

Recovered Image

✓Arithmetic Coding

•*Arithmetic coding* transforms input data into a single floating point number between 0 and 1

•There is not a direct correspondence between the code and the individual pixel values

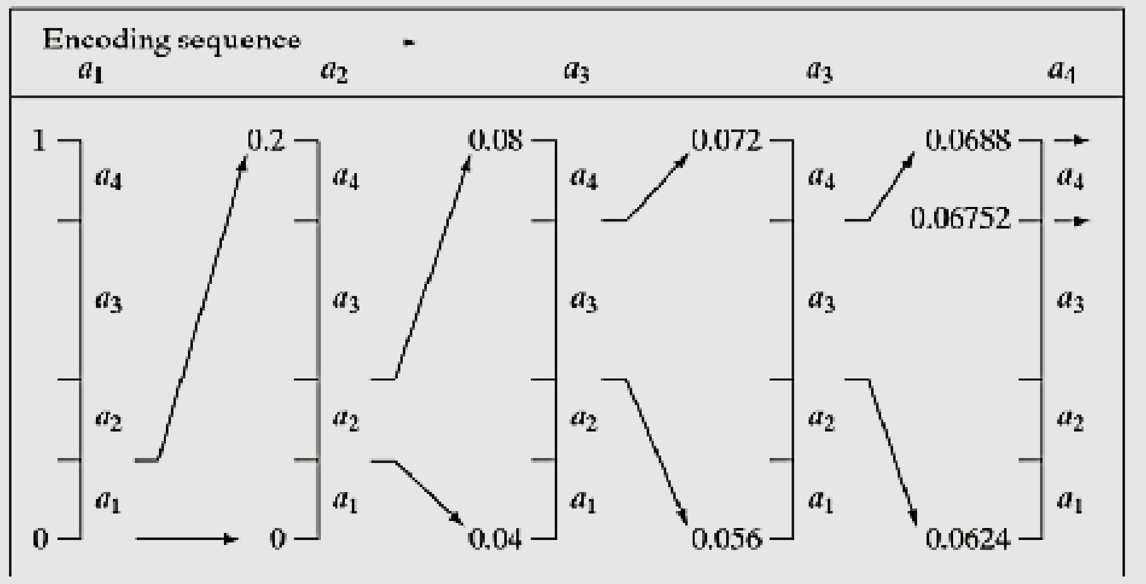
•As each input symbol (pixel value) is read the precision required for the number becomes greater

•As the images are very large and the precision of digital computers is finite, the entire image must be divided into small sub images to be encoded

•Arithmetic coding uses the probability distribution of the data (histogram), so it can theoretically achieve the maximum compression specified by the entropy

•It works by successively subdividing the interval between 0 and 1, based on the placement of the current pixel value in the probability distribution

ARITHMETIC CODING



Source Symbol	Probability	Initial Subinterval
a_1	0.2	[0.0, 0.2)
a_2	0.2	[0.2, 0.4)
a_3	0.4	[0.4, 0.8)
a_4	0.2	[0.8, 1.0)

✓Run-Length Coding

•Run-length coding (RLC) works by counting adjacent pixels with the same gray level value called the *run-length,* which is then encoded and stored

•RLC works best for binary, two-valued, images

•RLC can also work with complex images that have been preprocessed by thresholding to reduce the number of gray levels to two

•RLC can be implemented in various ways, but the first step is to define the required parameters

•Horizontal RLC (counting along the rows) or vertical RLC (counting along the columns) can be used

•In basic horizontal RLC, the number of bits used for the encoding depends on the number of pixels in a row

•If the row has 2^n pixels, then the required number of bits is *n*, so that a run that is the length of the entire row can be encoded

Run-Length Coding

$$
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

The RLC numbers are:

First row: 8

Second row: 0, 4, 4

Third row: 1, 2, 5

Fourth row: 1, 5, 2

Fifth row: 1, 3, 2, 1, 1

Sixth row: 2, 1, 2, 2, 1

Seventh row: 0, 4, 1, 1, 2

Eighth row: 8