Colors and formats

In this document several image formats and color palettes are listed.
Also some examples of converting between these formats and RGB 24-Bit color.
Grayscale and color formats and also palettes related to old computers and systems...

Grayscale formats

1. **Monochrone 1-Bit**
2. **Grayscale 2-Bit**
3. **Grayscale 3-Bit**
4. **Grayscale 4-Bit**
5. **Grayscale 8-Bit**

Color formats

1. **RGB 3-Bit**
2. **RGBi 4-Bit**
3. **RGB 6-Bit**
4. **RGB 8-Bit (True color)**
5. **RGB 9-Bit**
6. **RGB 12-Bit**
7. **RGB 15-Bit**
8. **RGB 16-Bit (High Color)**
9. **RGB 24-Bit**
10. **RGBA 32-Bit**
11. **Web-safe 216, 8-Bit**
12. **EGA 4-Bit**
13. **CGA 4-Bit (RGBi)**

```
type
 pRgb = ^TRgb;
 TRgb = packed record b,g,r:byte end; //RGB 24-Bit
```

LUT's (lookup tables)

Lookup tables (arrays) can replace some calculations and speed up the process...

```
const
 LUT_2BIT_TO_8BIT:array[0..3] of byte=(0,$55,$AA,$FF);
 LUT_3BIT_TO_8BIT:array[0..7] of byte=(0,$24,$49,$6D,$92,$B6,$DB,$FF);
 LUT_4BIT_TO_8BIT:array[0..$F] of byte=(
  0,$11,$22,$33,$44,$55,$66,$77,$88,$99,$AA,$BB,$CC,$DD,$EE,$FF
 );
```

Find nearest color from a palette

Some formats has no exact algorithm using math to convert from rgb,-
and a LUT of rgb24 would take 256^3 (16.777.216 bytes) of space and memory.
In those cases another option is to match nearest colors from a palette...

```
//find nearest color from a palette
function nearestLutColor(pLut,q:pointer;uMax:word):word;
var n,c,v:longword;nr,ng,nb:byte;p:pRgb;
begin
```

```
  with p^ do c:=sqr(b-nb)+sqr(g-ng)+sqr(r-nr);
  if c<v then begin v:=c;result:=n end;
  inc(p);
 end;
end;
```

Monochrone 1-Bit
Each bit defines the color for a pixel (8 pixels per byte).
Total 2 different colors possible, in this case black and white.

ID	Color	Value	Name
0x00		000000	**Black**
0x01		FFFFFF	**White**

```
const //Monochrome 1-Bit lookup table (LUT)
 LUT_MONO_1BIT:array[boolean] of TRgb=( //[0..1] -> [0..255]
  (b:$00;g:$00;r:$00),    //00. #000000    MONO_BLACK
  (b:$FF;g:$FF;r:$FF)     //01. #FFFFFF    MONO_WHITE
 );
```

RGB 24-Bit to Monochrome 1-Bit
Method 1: First convert to Gray 8-Bit, then check if greater than 127 ($FF div 2).
Method 2: Sum R+G+B value, then check if value is greater than 382 ($FF*3 div 2).

```
function rgb24_to_mono(p:pointer):byte;
const BOOL_TO_BYTE:array[boolean] of byte=(0,1);
```

Monochrome 1-Bit to RGB 24-Bit

```
function mono_to_rgb24(mono:byte):TRgb;
begin
 result:=LUT_MONO_1BIT[mono<>0]; //a value of [0..1]
end;
```

Grayscale 2-Bit

Total 4 shades of gray possible (4 pixels per byte).

ID	Color	Value	Name
0x00		000000	**Black**
0x01		555555	**Dark gray**
0x02		AAAAAA	**Light gray**
0x03		FFFFFF	**White**

```
const //Grayscale 2-Bit lookup table (LUT)
 LUT_GRAY_2BIT:array[0..3] of TRgb=( //Grayscale 2-Bit, 4 colors
  (b:$00;g:$00;r:$00), //00. #000000   GRAY2_BLACK
  (b:$55;g:$55;r:$55), //01. #555555   GRAY2_DARK
  (b:$AA;g:$AA;r:$AA), //02. #AAAAAA   GRAY2_LIGHT
  (b:$FF;g:$FF;r:$FF)  //03. #FFFFFF   GRAY2_WHITE
 );
```

RGB 24-Bit to Grayscale 2-Bit

```
 result := gray8 shr 6; //shr 6: get rid of 6 bits
end;
```

Grayscale 2-Bit to RGB 24-Bit

```
function gray2_to_rgb24(gray2:byte):TRgb;
begin
 if gray2>3 then gray2:=0;
 result := LUT_GRAY_2BIT[gray2]; //a value of [0..3]
end;
```

Grayscale 3-Bit

Total 8 shades of gray possible (~2.67 pixels per byte).

ID	Color	Value	Name
0x00		000000	**Black**
0x01		242424	**Darkest**
0x02		494949	**Darker**
0x03		6D6D6D	**Dark**
0x04		929292	**Light**
0x05		B6B6B6	**Lighter**
0x06		DBDBDB	**Lightest**
0x07		FFFFFF	**White**

```
const //Grayscale 3-Bit lookup table (LUT)
 LUT_GRAY_3BIT:array[0..7] of TRgb=(
  (b:$00;g:$00;r:$00), //00. #000000   GRAY3_BLACK
  (b:$24;g:$24;r:$24), //01. #242424   GRAY3_DARKEST
  (b:$49;g:$49;r:$49), //02. #494949   GRAY3_DARKER
  (b:$6D;g:$6D;r:$6D), //03. #6D6D6D   GRAY3_DARK
  (b:$92;g:$92;r:$92), //04. #929292   GRAY3_LIGHT
  (b:$B6;g:$B6;r:$B6), //05. #B6B6B6   GRAY3_LIGHTER
  (b:$DB;g:$DB;r:$DB), //06. #DBDBDB   GRAY3_LIGHTEST
  (b:$FF;g:$FF;r:$FF)  //07. #FFFFFF   GRAY3_WHITE
 );
```

```
begin
 with pRgb(p)^ do gray8 := (r+g+b) div 3; //first convert to gray8
 result := gray8 shr 5; //shr 5: get rid of 5 bits
end;
```

Grayscale 3-Bit to RGB 24-Bit

```
function gray3_to_rgb24(gray3:byte):TRgb;
begin
 if gray3>7 then gray3:=0;
 result:=LUT_GRAY_3BIT[gray3]; //a value of [0..7]
end;
```

Grayscale 4-Bit

Total 16 shades of gray possible (2 pixels per byte).

ID	Color	Value	Name
0x00		000000	Black
0x01		111111	Darker 4
0x02		222222	Darker 3
0x03		333333	Darker 2
0x04		444444	Darker 1
0x05		555555	Dark
0x06		666666	Medium Low 2
0x07		777777	Medium Low 1
0x08		888888	Medium High 1

0x0D	DDDDDD	**Lighter 3**
0x0E	EEEEEE	**Lighter 4**
0x0F	FFFFFF	**White**

```
const //Grayscale 4-Bit lookup table (LUT)
 LUT_GRAY_4BIT:array[0..$F] of TRgb=( //Grayscale 4-Bit, 16 colors
  (b:$00;g:$00;r:$00), //00. #000000   GRAY4_BLACK
  (b:$11;g:$11;r:$11), //01. #111111   GRAY4_DARKER_4
  (b:$22;g:$22;r:$22), //02. #222222   GRAY4_DARKER_3
  (b:$33;g:$33;r:$33), //03. #333333   GRAY4_DARKER_2
  (b:$44;g:$44;r:$44), //04. #444444   GRAY4_DARKER_1
  (b:$55;g:$55;r:$55), //05. #555555   GRAY4_DARK
  (b:$66;g:$66;r:$66), //06. #666666   GRAY4_MEDIUM_LOW_2
  (b:$77;g:$77;r:$77), //07. #777777   GRAY4_MEDIUM_LOW_1
  (b:$88;g:$88;r:$88), //08. #888888   GRAY4_MEDIUM_HIGH_1
  (b:$99;g:$99;r:$99), //09. #999999   GRAY4_MEDIUM_HIGH_2
  (b:$AA;g:$AA;r:$AA), //10. #AAAAAA   GRAY4_LIGHT
  (b:$BB;g:$BB;r:$BB), //11. #BBBBBB   GRAY4_LIGHTER_1
  (b:$CC;g:$CC;r:$CC), //12. #CCCCCC   GRAY4_LIGHTER_2
  (b:$DD;g:$DD;r:$DD), //13. #DDDDDD   GRAY4_LIGHTER_3
  (b:$EE;g:$EE;r:$EE), //14. #EEEEEE   GRAY4_LIGHTER_4
  (b:$FF;g:$FF;r:$FF)  //15. #FFFFFF   GRAY4_WHITE
 );
```

RGB 24-Bit to Grayscale 4-Bit

```
function rgb24_to_gray4(p:pointer):byte;
var gray8:byte;
begin
 with pRgb(p)^ do gray8 := (r+g+b) div 3; //first convert to gray8
 result := gray8 shr 4; //shr 4: get rid of 4 bits
end;
```

Grayscale 4-Bit to RGB 24-Bit

```
function gray4_to_rgb24(gray4:byte):TRgb;
begin
```

Grayscale 8-Bit

Total 256 shades of gray possible (1 pixel per byte).

RGB 24-Bit to Grayscale 8-Bit

```
function rgb24_to_gray8(p:pointer):byte;
begin
 with pRgb(p)^ do result := (r+g+b) div 3;
end;
```

Grayscale 8-Bit to RGB 24-Bit

```
function gray8_to_rgb24(gray8:byte):TRgb;
begin
 result.r := gray8;
 result.g := gray8;
 result.b := gray8;
end;
```

RGB 3-Bit

Total 8 different colors possible (~2.67 pixels per byte).
A single bit for each R,G,B values (3 bits).

ID	Color	Value	Name
0x00		000000	**Black**
0x01		0000FF	**Blue**
0x02		00FF00	**Green**
0x03		00FFFF	**Cyan**
0x04		FF0000	**Red**
0x05		FF00FF	**Magenta**
0x06		FFFF00	**Yellow**
0x07		FFFFFF	**White**

```
const //RGB 3-Bit lookup table (LUT)
 LUT_RGB_3BIT:array[0..7] of TRgb=( //RGB 3-Bit, 8 colors.
  (b:$00;g:$00;r:$00), //00. #000000   RGB3_BLACK
  (b:$FF;g:$00;r:$00), //01. #0000FF   RGB3_BLUE
  (b:$00;g:$FF;r:$00), //02. #00FF00   RGB3_GREEN
  (b:$FF;g:$FF;r:$00), //03. #00FFFF   RGB3_CYAN
  (b:$00;g:$00;r:$FF), //04. #FF0000   RGB3_RED
  (b:$FF;g:$00;r:$FF), //05. #FF00FF   RGB3_MAGENTA
  (b:$00;g:$FF;r:$FF), //06. #FFFF00   RGB3_YELLOW
  (b:$FF;g:$FF;r:$FF)  //07. #FFFFFF   RGB3_WHITE
 );
```

RGB 24-Bit to RGB 3-Bit

```
function rgb24_to_rgb3(p:pointer):byte;
begin
 with pRgb(p)^ do
  result:=(r shr 7) shl 2  //shl 2: put at 3rd bit.
       or (g shr 7) shl 1  //shl 1: put at 2nd bit.
       or (b shr 7);       //shr 7: get rid of 7 bits
end;
```

```
begin
 if rgb3>7 then rgb3:=0;
 result:=LUT_RGB_3BIT[rgb3]; //a value of [0..7]
end;
```

RGBi 4-Bit
Total 16 different colors possible (2 pixels per byte).
A single bit for each R,G,B values + I intensity (4 bits).

ID	Color	Value	Name
0x00		000000	**Black**
0x01		000091	**Dark blue**
0x02		009100	**Dark green**
0x03		009191	**Dark cyan**
0x04		910000	**Dark red**
0x05		910091	**Dark magenta**
0x06		919100	**Dark yellow**
0x07		B7B7B7	**Gray**
0x08		686868	**Dark gray**
0x09		0000FF	**Blue**
0x0A		00FF00	**Green**
0x0B		00FFFF	**Cyan**
0x0C		FF0000	**Red**
0x0D		FF00FF	**Magenta**
0x0E		FFFF00	**Yellow**
0x0F		FFFFFF	**White**

```
  (b:$91;g:$00;r:$00), //01. #000091   RGBI_BLUE_DARK
  (b:$00;g:$91;r:$00), //02. #009100   RGBI_GREEN_DARK
  (b:$91;g:$91;r:$00), //03. #009191   RGBI_CYAN_DARK
  (b:$00;g:$00;r:$91), //04. #910000   RGBI_RED_DARK
  (b:$91;g:$00;r:$91), //05. #910091   RGBI_MAGENTA_DARK
  (b:$00;g:$91;r:$91), //06. #919100   RGBI_YELLOW_DARK
  (b:$B7;g:$B7;r:$B7), //07. #B7B7B7   RGBI_GRAY_LIGHT
  (b:$68;g:$68;r:$68), //08. #686868   RGBI_GRAY_DARK
  (b:$FF;g:$00;r:$00), //09. #0000FF   RGBI_BLUE
  (b:$00;g:$FF;r:$00), //10. #00FF00   RGBI_GREEN
  (b:$FF;g:$FF;r:$00), //11. #00FFFF   RGBI_CYAN
  (b:$00;g:$00;r:$FF), //12. #FF0000   RGBI_RED
  (b:$FF;g:$00;r:$FF), //13. #FF00FF   RGBI_MAGENTA
  (b:$00;g:$FF;r:$FF), //14. #FFFF00   RGBI_YELLOW
  (b:$FF;g:$FF;r:$FF)  //15. #FFFFFF   RGBI_WHITE
 );
```

RGB 24-Bit to RGBi 4-Bit

```
function rgb24_to_rgbi(p:pointer;scale:word=300):byte; //[0..15]
var bl:boolean;
begin
 with pRgb(p)^ do
  result:=byte(r+g+b>scale) shl 3   //shl 3: put at 4th bit.
       or (r shr 7) shl 2           //shl 2: put at 3rd bit.
       or (g shr 7) shl 1           //shl 1: put at 2nd bit.
       or (b shr 7);                //shr 7: get rid of 7 bits
end;
```

RGBi 4-Bit to RGB 24-Bit

```
function rgbi_to_rgb24(rgbi:byte):TRgb;
begin
 if rgbi>$F then rgbi:=0;
 result:=LUT_RGBI_4BIT[rgbi]; //a value of [0..15]
end;
```

0x00	000000	0x10	550000	0x20	AA0000	0x30	FF0000
0x01	000055	0x11	550055	0x21	AA0055	0x31	FF0055
0x02	0000AA	0x12	5500AA	0x22	AA00AA	0x32	FF00AA
0x03	0000FF	0x13	5500FF	0x23	AA00FF	0x33	FF00FF
0x04	005500	0x14	555500	0x24	AA5500	0x34	FF5500
0x05	005555	0x15	555555	0x25	AA5555	0x35	FF5555
0x06	0055AA	0x16	5555AA	0x26	AA55AA	0x36	FF55AA
0x07	0055FF	0x17	5555FF	0x27	AA55FF	0x37	FF55FF
0x08	00AA00	0x18	55AA00	0x28	AAAA00	0x38	FFAA00
0x09	00AA55	0x19	55AA55	0x29	AAAA55	0x39	FFAA55
0x0A	00AAAA	0x1A	55AAAA	0x2A	AAAAAA	0x3A	FFAAAA
0x0B	00AAFF	0x1B	55AAFF	0x2B	AAAAFF	0x3B	FFAAFF
0x0C	00FF00	0x1C	55FF00	0x2C	AAFF00	0x3C	FFFF00
0x0D	00FF55	0x1D	55FF55	0x2D	AAFF55	0x3D	FFFF55
0x0E	00FFAA	0x1E	55FFAA	0x2E	AAFFAA	0x3E	FFFFAA
0x0F	00FFFF	0x1F	55FFFF	0x2F	AAFFFF	0x3F	FFFFFF

```
const //RGB 6-Bit lookup table (LUT)
 LUT_RGB_6BIT:array[0..63] of TRgb=( //RGB 6-Bit, 64 colors.
  (b:$00;g:$00;r:$00),(b:$55;g:$00;r:$00),(b:$AA;g:$00;r:$00),
  (b:$FF;g:$00;r:$00),(b:$00;g:$55;r:$00),(b:$55;g:$55;r:$00),
  (b:$AA;g:$55;r:$00),(b:$FF;g:$55;r:$00),(b:$00;g:$AA;r:$00),
  (b:$55;g:$AA;r:$00),(b:$AA;g:$AA;r:$00),(b:$FF;g:$AA;r:$00),
  (b:$00;g:$FF;r:$00),(b:$55;g:$FF;r:$00),(b:$AA;g:$FF;r:$00),
  (b:$FF;g:$FF;r:$00),(b:$00;g:$00;r:$55),(b:$55;g:$00;r:$55),
  (b:$AA;g:$00;r:$55),(b:$FF;g:$00;r:$55),(b:$00;g:$55;r:$55),
  (b:$55;g:$55;r:$55),(b:$AA;g:$55;r:$55),(b:$FF;g:$55;r:$55),
```

```
  (b:$00;g:$55;r:$AA),(b:$55;g:$55;r:$AA),(b:$AA;g:$55;r:$AA),
  (b:$FF;g:$55;r:$AA),(b:$00;g:$AA;r:$AA),(b:$55;g:$AA;r:$AA),
  (b:$AA;g:$AA;r:$AA),(b:$FF;g:$AA;r:$AA),(b:$00;g:$FF;r:$AA),
  (b:$55;g:$FF;r:$AA),(b:$AA;g:$FF;r:$AA),(b:$FF;g:$FF;r:$AA),
  (b:$00;g:$00;r:$FF),(b:$55;g:$00;r:$FF),(b:$AA;g:$00;r:$FF),
  (b:$FF;g:$00;r:$FF),(b:$00;g:$55;r:$FF),(b:$55;g:$55;r:$FF),
  (b:$AA;g:$55;r:$FF),(b:$FF;g:$55;r:$FF),(b:$00;g:$AA;r:$FF),
  (b:$55;g:$AA;r:$FF),(b:$AA;g:$AA;r:$FF),(b:$FF;g:$AA;r:$FF),
  (b:$00;g:$FF;r:$FF),(b:$55;g:$FF;r:$FF),(b:$AA;g:$FF;r:$FF),
  (b:$FF;g:$FF;r:$FF)
 );
```

RGB 24-Bit to RGB 6-Bit

```
function rgb24_to_rgb6(p:pointer):byte;
begin
 with pRgb(p)^ do
  result:=(r shr 6) shl 4  //shl 4: put at 5th bit.
       or (g shr 6) shl 2  //shl 2: put at 3rd bit.
       or (b shr 6);  //shr 6: get rid of 6 bits
end;
```

RGB 6-Bit to RGB 24-Bit

```
//conversion using a single LUT for the palette
function rgb6_to_rgb24(rgb6:byte):TRgb;
begin
 if rgb6>63 then rgb6:=0;
 result:=LUT_RGB_6BIT[rgb6]; //a value of [0..63]
end;

//OR:

//conversion of each RGB value from 2-Bit to 8-Bit
procedure rgb6_to_rgb24_alt(rgb6:byte):TRgb;
begin
 result.r:=LUT_2BIT_TO_8BIT[rgb6 shr 4 and 3]; //shr 4: get rid of 5 bits
 result.g:=LUT_2BIT_TO_8BIT[rgb6 shr 2 and 3]; //and 3: keep only 2 bits
```

RGB 8-Bit (True color)

Total 256 different colors possible (1 pixel per byte).
3 bits of red, 3 bits of green, 2 bits of blue. (RRRGGGBB)

RGB 24-Bit to RGB 8-Bit

```
function rgb24_to_rgb8(p:pointer):byte;
begin
 with pRgb(p)^ do
  result:=(r shr 5) shl 5  //shl 5: put at 6th bit.
       or (g shr 5) shl 2  //shl 2: put at 3th bit.
       or (b shr 6);       //shr 6: get rid of 6 bits
end;
```

RGB 8-Bit to RGB 24-Bit

```
function rgb8_to_rgb24(rgb8:byte):TRgb;
begin
 result.r := LUT_3BIT_TO_8BIT[rgb8 shr 5];        //Red:   3 bits
 result.g := LUT_3BIT_TO_8BIT[rgb8 shr 2 and 7]; //Green: 3 bits
 result.b := LUT_2BIT_TO_8BIT[rgb8 and 3];        //Blue:  2 bits
end;
```

RGB 9-Bit

Total 512 different colors possible.
Three bits for each R,G,B values (9 bits).

RGB 24-Bit to RGB 9-Bit

```
function rgb24_to_rgb9(p:pointer):word; //8*8*8 = 512
begin
 with pRgb(p)^ do
  result:=(r shr 5) shl 6    //shl 6: put at 7th bit.
       or (g shr 5) shl 3    //shl 3: put at 4th bit.
       or (b shr 5);         //shr 5: get rid of 5 bits
end;
```

RGB 9-Bit to RGB 24-Bit

```
function rgb9_to_rgb24(rgb9:word):TRgb;
begin
 result.r := LUT_3BIT_TO_8BIT[rgb9 shr 6 and 7]; //shr 6: get rid of 6 bits
 result.g := LUT_3BIT_TO_8BIT[rgb9 shr 3 and 7]; //and 7: keep only 3 bits
 result.b := LUT_3BIT_TO_8BIT[rgb9 and 7]; //and 7: keep only 3 bits
end;
```

RGB 12-Bit

Total 4096 different colors possible.
Four bits for each R,G,B values (12 bits).

```
 with pRgb(p)^ do
  result:=(r shr 4) shl 8  //shl 8: put at 9th bit.
       or (g shr 4) shl 4  //shl 4: put at 5th bit.
       or (b shr 4);       //shr 4: get rid of 4 bits
end;
```

RGB 12-Bit to RGB 24-Bit

```
function rgb12_to_rgb24(n:word):TRgb;
begin
 result.r := LUT_4BIT_TO_8BIT[n shr 8 and $F]; //shr 8: get rid of 8 bits
 result.g := LUT_4BIT_TO_8BIT[n shr 4 and $F]; //and $F: keep only 4 bits
 result.b := LUT_4BIT_TO_8BIT[n and $F]; //and $F: keep only 4 bits
end;
```

RGB 15-Bit

Total 32768 different colors possible.
Five bits for each R,G,B values (15 bits).

RGB 24-Bit to RGB 15-Bit

```
function rgb24_to_rgb15(p:pointer):word;
begin
 with pRgb(p)^ do
  result:=(r shr 3) shl 10  //shl 10: put at 11th bit.
       or (g shr 3) shl 5   //shl 5: put at 6th bit.
       or (b shr 3);        //shr 3: get rid of 3 bits
end;
```

RGB 15-Bit to RGB 24-Bit

```
function rgb15_to_rgb24(rgb15:word):TRgb;
begin
result.b := (rgb15 and $1F) shl 3;
result.g := ((rgb15 shr 5) and $1F) shl 3;
result.r := ((rgb15 shr 10) and $1F) shl 3;
end;
```

RGB 24-Bit to RGB 16-Bit

```
function rgb24_to_rgb16(p:pointer):word;
begin
 with pRgb(p)^ do
  result:=(r shr 3) shl 11  //shl 11: put at 12th bit.
       or (g shr 2) shl 5   //shl 5: put at 6th bit.
       or (b shr 3);        //shr 3: get rid of 3 bits
end;
```

RGB 16-Bit to RGB 24-Bit

```
function rgb16_to_rgb24(rgb16:word):TRgb;
begin
 result.r := ((rgb16 and $F800) shr 11) shl 3;
 result.g := ((rgb16 and $07E0) shr 5) shl 2;
 result.b := (rgb16 and $1F) shl 3;
end;
```

RGB 24-Bit

A byte (8 bits) for each value of R,G,B (24 bits).
Total 16.777.216 different colors possible.

RGBA 32-Bit

A byte (8 bits) for each value of R,G,B (24 bits).
Also a byte for the alpha channel (transparency).
Four bytes per pixel (32 bits).

Total 216 colors possible.

RGB 24-Bit to Web216 8-Bit

```
function rgb24_to_web216(p:pointer):byte;
begin
 with pRgb(p)^ do
  result:=b div $33*$24
        +(g div $33*6)
        +(r div $33);
end;
```

Web216 8-Bit to RGB 24-Bit

```
function web216_to_rgb24(n:byte):TRgb;
begin
 if n>$D7 then n:=$D7;
 result.b := n div $24*$33;n:=n mod $24;
 result.g := n div 6*$33;n:=n mod 6;
 result.r := n*$33;
end;
```

EGA 4-Bit

Total 16 colors possible (2 pixels per byte).

ID	Color	Value	Name
0x00		000000	**Black**
0x01		680000	**Darker red**
0x02		006800	**Dark green**
0x03		B70000	**Dark red**
0x04		686800	**Dark yellow**
0x05		FF0000	**Red**
0x06		686868	**Dark gray**
0x07		6868B7	**Blue**
0x08		B76800	**Brown**
0x09		B76868	**Light red**
0x0A		B768B7	**Purple**
0x0B		68B700	**Light green**
0x0C		FF6800	**Orange**
0x0D		B7B7B7	**Light gray**
0x0E		FFB768	**Yellow**
0x0F		FFFFFF	**White**

```
const //EGA 4-Bit lookup table (LUT)
 LUT_EGA_4BIT:array[0..$F] of TRgb=( //EGA 4-Bit, 16 colors
  (b:$00;g:$00;r:$00), //00. #000000   EGA4_BLACK
  (b:$00;g:$00;r:$68), //01. #680000   EGA4_RED_DARKER
  (b:$00;g:$68;r:$00), //02. #006800   EGA4_GREEN_DARK
  (b:$00;g:$00;r:$B7), //03. #B70000   EGA4_RED_DARK
  (b:$00;g:$68;r:$68), //04. #686800   EGA4_YELLOW_DARK
  (b:$00;g:$00;r:$FF), //05. #FF0000   EGA4_RED
  (b:$68;g:$68;r:$68), //06. #686868   EGA4_GRAY_DARK
  (b:$B7;g:$68;r:$68), //07. #6868B7   EGA4_BLUE
  (b:$00;g:$68;r:$B7), //08. #B76800   EGA4_BROWN
  (b:$68;g:$68;r:$B7), //09. #B76868   EGA4_RED_LIGHT
```

```
  (b:$68;g:$B7;r:$FF), //14. #FFB768   EGA4_YELLOW
  (b:$FF;g:$FF;r:$FF)  //15. #FFFFFF   EGA4_WHITE
 );
```

RGB 24-Bit to EGA 4-Bit

```
//Match nearest colors from the palette.
function rgb24_to_ega4(p:pointer):byte;
var palette:pointer;maxid:byte;
begin
 palette := @LUT_EGA_4BIT[0];
 maxid   := $F;
 result  := nearestLutColor(palette,p,maxid); //see above
end;
```

EGA 4-Bit to RGB 24-Bit

```
function ega4_to_rgb24(ega4:byte):TRgb;
begin
 if ega4>$F then ega4:=0;
 result:=LUT_EGA_4BIT[ega4]; //a value of [0..15]
end;
```

CGA 4-Bit (RGBi)

Total 16 colors possible (2 pixels per byte).
Based on RGBi as described above.

0x03	00AAAA	**Cyan low**
0x04	AA0000	**Red low**
0x05	AA00AA	**Magenta low**
0x06	AA5500	**Brown**
0x07	AAAAAA	**Gray high**
0x08	555555	**Gray low**
0x09	5555FF	**Blue high**
0x0A	55FF55	**Green high**
0x0B	55FFFF	**Cyan high**
0x0C	FF5555	**Red high**
0x0D	FF55FF	**Magenta high**
0x0E	FFFF55	**Yellow**
0x0F	FFFFFF	**White**

```
const //CGA 4-Bit (RGBi) lookup table (LUT)
 LUT_CGA_RGBI_4BIT:array[0..$F] of TRgb=( //CGA RGBI 4-Bit, 16 colors
  (b:$00;g:$00;r:$00), //00. #000000   CGA4_BLACK
  (b:$AA;g:$00;r:$00), //01. #0000AA   CGA4_BLUE_LOW
  (b:$00;g:$AA;r:$00), //02. #00AA00   CGA4_GREEN_LOW
  (b:$AA;g:$AA;r:$00), //03. #00AAAA   CGA4_CYAN_LOW
  (b:$00;g:$00;r:$AA), //04. #AA0000   CGA4_RED_LOW
  (b:$AA;g:$00;r:$AA), //05. #AA00AA   CGA4_MAGENTA_LOW
  (b:$00;g:$55;r:$AA), //06. #AA5500   CGA4_BROWN
  (b:$AA;g:$AA;r:$AA), //07. #AAAAAA   CGA4_GRAY_LIGHT
  (b:$55;g:$55;r:$55), //08. #555555   CGA4_GRAY_DARK
  (b:$FF;g:$55;r:$55), //09. #5555FF   CGA4_BLUE_HIGH
  (b:$55;g:$FF;r:$55), //10. #55FF55   CGA4_GREEN_HIGH
  (b:$FF;g:$FF;r:$55), //11. #55FFFF   CGA4_CYAN_HIGH
  (b:$55;g:$55;r:$FF), //12. #FF5555   CGA4_RED_HIGH
  (b:$FF;g:$55;r:$FF), //13. #FF55FF   CGA4_MAGENTA_HIGH
  (b:$55;g:$FF;r:$FF), //14. #FFFF55   CGA4_YELLOW
  (b:$FF;g:$FF;r:$FF)  //15. #FFFFFF   CGA4_WHITE
 );
```

RGB 24-Bit to CGA 4-Bit

```
end;
```

CGA 4-Bit to RGB 24-Bit

```
function cga4_to_rgb24(cga4:byte):TRgb;
begin
 if cga4>$F then cga4:=0;
 result:=LUT_CGA_RGBI_4BIT[cga4]; //a value of [0..15]
end;
```

CGA 2-Bit (Mode 5)

Total 4 colors possible (4 pixels per byte).

ID	Color	Value	Name
0x00		000000	**Black**
0x01		55FF55	**Green bright**
0x02		FF5555	**Red bright**
0x03		FFFF55	**Yellow**

```
const //CGA 2-Bit Mode 5 lookup table (LUT)
 LUT_CGA_2BIT_MODE5:array[0..3] of TRgb=( //CGA 2-Bit A, 4 colors.
  (b:$00;g:$00;r:$00), //00. #000000   CGA2A_BLACK
  (b:$55;g:$FF;r:$55), //01. #55FF55   CGA2A_GREEN_BRIGHT
  (b:$55;g:$55;r:$FF), //02. #FF5555   CGA2A_RED_BRIGHT
  (b:$55;g:$FF;r:$FF)  //03. #FFFF55   CGA2A_YELLOW
 );
```

```
function rgb24_to_cga2(p:pointer):byte;
var cga4:byte;
begin
 cga4:=rgb24_to_cga4(p); //first convert to cga 4-Bit
 result:=CGA4_TO_CGA2[cga4]; //then pick related color from array
end;
```

CGA 2-Bit Mode 5 to RGB 24-Bit

```
function cga2_to_rgb24(cga2:byte):TRgb;
begin
 if cga2>3 then cga2:=0;
 result:=LUT_CGA_2BIT_MODE5[cga2]; //a value of [0..3]
end;
```

Windows 4-Bit

Total 16 colors possible (2 pixels per byte).
The 16 default color palette from Microsoft Windows.
RGBi variation, colors in different order.

ID	Color	Value	Name
0x00		000000	**Black**
0x01		800000	**Maroon**
0x02		008000	**Green**
0x03		808000	**Olive**
0x04		000080	**Navy**
0x05		800080	**Purple**

0x09		FF0000	Red
0x0A		00FF00	Lime
0x0B		FFFF00	Yellow
0x0C		0000FF	Blue
0x0D		FF00FF	Fuchsia
0x0E		00FFFF	Aqua
0x0F		FFFFFF	White

```
const //MSWIN 4-Bit lookup table (LUT)
 LUT_MSWIN_4BIT:array[0..$F] of TRgb=( //16 default color palette (4-Bit)
  (b:$00;g:$00;r:$00), //00. #000000   WIN_BLACK
  (b:$00;g:$00;r:$80), //01. #800000   WIN_MAROON
  (b:$00;g:$80;r:$00), //02. #008000   WIN_GREEN
  (b:$00;g:$80;r:$80), //03. #808000   WIN_OLIVE
  (b:$80;g:$00;r:$00), //04. #000080   WIN_NAVY
  (b:$80;g:$00;r:$80), //05. #800080   WIN_PURPLE
  (b:$80;g:$80;r:$00), //06. #008080   WIN_TEAL
  (b:$C0;g:$C0;r:$C0), //07. #C0C0C0   WIN_SILVER
  (b:$80;g:$80;r:$80), //08. #808080   WIN_GRAY
  (b:$00;g:$00;r:$FF), //09. #FF0000   WIN_RED
  (b:$00;g:$FF;r:$00), //10. #00FF00   WIN_LIME
  (b:$00;g:$FF;r:$FF), //11. #FFFF00   WIN_YELLOW
  (b:$FF;g:$00;r:$00), //12. #0000FF   WIN_BLUE
  (b:$FF;g:$00;r:$FF), //13. #FF00FF   WIN_FUCHSIA
  (b:$FF;g:$FF;r:$00), //14. #00FFFF   WIN_AQUA
  (b:$FF;g:$FF;r:$FF)  //15. #FFFFFF   WIN_WHITE
 );
```

RGB 24-Bit to Windows 4-Bit

```
const
 RGBI_TO_MSWIN4:array[0..$F] of byte=(0,4,2,6,1,5,3,7,8,12,10,14,9,13,11,15);

function rgb24_to_mswin4(p:pointer):byte;
var rgbi:byte;
begin
 rgbi:=rgb24_to_rgbi(p); //first convert to RGBi (4-Bit)
```

Windows 4-Bit to RGB 24-Bit

```
function mswin4_to_rgb24(win4:byte):TRgb;
begin
 if win4>$F then win4:=0;
 result:=LUT_MSWIN_4BIT[win4]; //a value of [0..15]
end;
```

Apple II 4-Bit

Total 16 colors possible (2 pixels per byte).
A duplicate gray results in 15 colors instead of 16...

ID	Color	Value	Name
0x00		000000	**Black**
0x01		722640	**Magenta**
0x02		40337F	**Dark blue**
0x03		E434FE	**Purple**
0x04		0E5940	**Dark green**
0x05		808080	**Gray**
0x06		1B9AFE	**Medium blue**
0x07		BFB3FF	**Light blue**
0x08		404C00	**Brown**
0x09		E46501	**Orange**
0x0A		808080	**Duplicate gray**
0x0B		F1A6BF	**Pink**
0x0C		1BCB01	**Green**
0x0D		BFCC80	**Yellow**

```
const //Apple II 4-Bit lookup table (LUT)
 LUT_APPLEII_4BIT:array[0..$F] of TRgb=( //Apple II, 4-Bit
  (b:$00;g:$00;r:$00), //00. #000000   APPLE_II_BLACK
  (b:$40;g:$26;r:$72), //01. #722640   APPLE_II_MAGENTA
  (b:$7F;g:$33;r:$40), //02. #40337F   APPLE_II_BLUE_DARK
  (b:$FE;g:$34;r:$E4), //03. #E434FE   APPLE_II_PURPLE
  (b:$40;g:$59;r:$0E), //04. #0E5940   APPLE_II_GREEN_DARK
  (b:$80;g:$80;r:$80), //05. #808080   APPLE_II_GRAY
  (b:$FE;g:$9A;r:$1B), //06. #1B9AFE   APPLE_II_BLUE_MEDIUM
  (b:$FF;g:$B3;r:$BF), //07. #BFB3FF   APPLE_II_BLUE_LIGHT
  (b:$00;g:$4C;r:$40), //08. #404C00   APPLE_II_BROWN
  (b:$01;g:$65;r:$E4), //09. #E46501   APPLE_II_ORANGE
  (b:$80;g:$80;r:$80), //10. #808080   APPLE_II_GRAY_DUPL
  (b:$BF;g:$A6;r:$F1), //11. #F1A6BF   APPLE_II_PINK
  (b:$01;g:$CB;r:$1B), //12. #1BCB01   APPLE_II_GREEN
  (b:$80;g:$CC;r:$BF), //13. #BFCC80   APPLE_II_YELLOW
  (b:$BF;g:$D9;r:$8D), //14. #8DD9BF   APPLE_II_AQUA
  (b:$FF;g:$FF;r:$FF)  //15. #FFFFFF   APPLE_II_WHITE
 );
```

RGB 24-Bit to Apple II 4-Bit

```
//Match nearest color from the palette...
function rgb24_to_appleii(p:pointer):byte;
var palette:pointer;maxid:byte;
begin
 palette := @LUT_APPLEII_4BIT[0];
 maxid   := $F;
 result  := nearestLutColor(palette,p,maxid); //see above
end;
```

Apple II 4-Bit to RGB 24-Bit

```
function appleii4_to_rgb24(n:byte):TRgb;
begin
 if n>$F then n:=0;
 result:=LUT_APPLEII_4BIT[n]; //a value of [0..15]
end;
```

ID	Color	Value	Name
0x00		000000	**Black**
0x01		FFFFFF	**White**
0x02		8D3E37	**Red**
0x03		72C1C8	**Cyan**
0x04		80348B	**Purple**
0x05		55A049	**Green**
0x06		40318D	**Blue**
0x07		AAB95D	**Yellow**
0x08		8B5429	**Orange**
0x09		D59F74	**Light orange**
0x0A		B86962	**Light red**
0x0B		87D6DD	**Light cyan**
0x0C		AA5FB6	**Light purple**
0x0D		94E089	**Light green**
0x0E		8071CC	**Light blue**
0x0F		BFCE72	**Light yellow**

```
const //Commodore VIC-20 4-Bit lookup table (LUT)
 LUT_VIC20_4BIT:array[0..$F] of TRgb=( //VIC20 4-Bit, 16 colors
  (b:$00;g:$00;r:$00), //00. #000000   VIC20_BLACK
  (b:$FF;g:$FF;r:$FF), //01. #FFFFFF   VIC20_WHITE
  (b:$37;g:$3E;r:$8D), //02. #8D3E37   VIC20_RED
  (b:$C8;g:$C1;r:$72), //03. #72C1C8   VIC20_CYAN
  (b:$8B;g:$34;r:$80), //04. #80348B   VIC20_PURPLE
```

```
  (b:$74;g:$9F;r:$D5), //09. #D59F74   VIC20_ORANGE_LIGHT
  (b:$62;g:$69;r:$B8), //10. #B86962   VIC20_RED_LIGHT
  (b:$DD;g:$D6;r:$87), //11. #87D6DD   VIC20_CYAN_LIGHT
  (b:$B6;g:$5F;r:$AA), //12. #AA5FB6   VIC20_PURPLE_LIGHT
  (b:$89;g:$E0;r:$94), //13. #94E089   VIC20_GREEN_LIGHT
  (b:$CC;g:$71;r:$80), //14. #8071CC   VIC20_BLUE_LIGHT
  (b:$72;g:$CE;r:$BF)  //15. #BFCE72   VIC20_YELLOW_LIGHT
 );
```

RGB 24-Bit to Vic-20 4-Bit

```
//Match nearest color from the palette...
function rgb24_to_vic20(p:pointer):byte;
var palette:pointer;maxid:byte;
begin
 palette := @LUT_VIC20_4BIT[0];
 maxid   := $F;
 result  := nearestLutColor(palette,p,maxid); //see above
end;
```

Vic-20 4-Bit to RGB 24-Bit

```
function vic20_to_rgb24(vic20:byte):TRgb;
begin
 if vic20>$F then vic20:=0;
 result:=LUT_VIC20_4BIT[vic20]; //a value of [0..15]
end;
```

Commodore 64, 4-Bit

Commodore 64 palette, 4-Bit.
Total 16 colors possible (2 pixels per byte).

ID	Color	Value	Name
0x00		000000	**Black**
0x01		FFFFFF	**White**
0x02		883932	**Red**
0x03		67B6BD	**Cyan**
0x04		8B3F96	**Purple**
0x05		55A049	**Green**
0x06		40318D	**Blue**
0x07		BFCE72	**Yellow**
0x08		8B5429	**Orange**
0x09		574200	**Brown**
0x0A		B86962	**Light red**
0x0B		505050	**Dark gray**
0x0C		787878	**Gray**
0x0D		94E089	**Light green**
0x0E		7869C4	**Light blue**
0x0F		9F9F9F	**Light gray**

```
const //Commodore 64 4-Bit lookup table (LUT)
 LUT_C64_4BIT:array[0..$F] of TRgb=( //C64 4-Bit, 16 colors.
  (b:$00;g:$00;r:$00), //00. #000000   C64_BLACK
  (b:$FF;g:$FF;r:$FF), //01. #FFFFFF   C64_WHITE
  (b:$32;g:$39;r:$88), //02. #883932   C64_RED
  (b:$BD;g:$B6;r:$67), //03. #67B6BD   C64_CYAN
  (b:$96;g:$3F;r:$8B), //04. #8B3F96   C64_PURPLE
  (b:$49;g:$A0;r:$55), //05. #55A049   C64_GREEN
  (b:$8D;g:$31;r:$40), //06. #40318D   C64_BLUE
  (b:$72;g:$CE;r:$BF), //07. #BFCE72   C64_YELLOW
  (b:$29;g:$54;r:$8B), //08. #8B5429   C64_ORANGE
  (b:$00;g:$42;r:$57), //09. #574200   C64_BROWN
```

```
  (b:$C4;g:$69;r:$78), //14. #7869C4   C64_BLUE_LIGHT
  (b:$9F;g:$9F;r:$9F)  //15. #9F9F9F   C64_GRAY_LIGHT
 );
```

RGB 24-Bit to C64 4-Bit

```
//Match nearest color from the palette...
function rgb24_to_c64(p:pointer):byte;
var palette:pointer;maxid:byte;
begin
 palette := @LUT_C64_4BIT[0];
 maxid   := $F;
 result  := nearestLutColor(palette,p,maxid); //see above
end;
```

C64 4-Bit to RGB 24-Bit

```
function c64_to_rgb24(c64:byte):TRgb;
begin
 if c64>$F then c64:=0;
 result:=LUT_C64_4BIT[c64]; //a value of [0..15]
end;
```

Commodore Amiga 2-Bit

Commodore Amiga 2-Bit palette.
Total 4 different colors possible (4 pixels per byte).

0x03 EECC99 **Yellow**

```
const //Commodore Amiga 2-Bit lookup table (LUT)
 LUT_AMIGA_2BIT:array[0..3] of TRgb=( //Amiga 2-Bit, 4 colors
  (b:$00;g:$00;r:$00), //00. #000000   AMIGA_2BIT_BLACK
  (b:$22;g:$33;r:$EE), //01. #EE3322   AMIGA_2BIT_RED
  (b:$55;g:$66;r:$66), //02. #666655   AMIGA_2BIT_GRAY
  (b:$99;g:$CC;r:$EE)  //03. #EECC99   AMIGA_2BIT_YELLOW
 );
```

RGB 24-Bit to Amiga 2-Bit

```
//Match nearest color from the palette...
function rgb24_to_amiga2(p:pointer):byte;
var palette:pointer;maxid:byte;
begin
 palette := @LUT_AMIGA_2BIT[0];
 maxid   := 3;
 result  := nearestLutColor(palette,p,maxid); //see above
end;
```

Amiga 2-Bit to RGB 24-Bit

```
function amiga2_to_rgb24(amiga2:byte):TRgb;
begin
 if amiga2>3 then amiga2:=0;
 result:=LUT_AMIGA_2BIT[amiga2]; //a value of [0..3]
end;
```

Commodore Amiga 3-Bit

Commodore Amiga 3-Bit palette.
Total 8 different colors possible (~2.67 pixels per byte).

ID	Color	Value	Name
0x00		000000	**Black**
0x01		A80000	**Dark red**
0x02		447800	**Green**
0x03		FF5631	**Light red**
0x04		788998	**Gray**
0x05		FF8956	**Orange**
0x06		FFC678	**Yellow**
0x07		F1F1D4	**White**

```
const //Commodore Amiga 3-Bit lookup table (LUT)
 LUT_AMIGA_3BIT:array[0..7] of TRgb=( //Amiga 3-Bit, 8 colors
  (b:$00;g:$00;r:$00), //00. #000000   AMIGA_3BIT_BLACK
  (b:$00;g:$00;r:$A8), //01. #A80000   AMIGA_3BIT_RED_DARK
  (b:$00;g:$78;r:$44), //02. #447800   AMIGA_3BIT_GREEN
  (b:$31;g:$56;r:$FF), //03. #FF5631   AMIGA_3BIT_RED_LIGHT
  (b:$98;g:$89;r:$78), //04. #788998   AMIGA_3BIT_GRAY
  (b:$56;g:$89;r:$FF), //05. #FF8956   AMIGA_3BIT_ORANGE
  (b:$78;g:$C6;r:$FF), //06. #FFC678   AMIGA_3BIT_YELLOW
  (b:$D4;g:$F1;r:$F1)  //07. #F1F1D4   AMIGA_3BIT_WHITE
 );
```

RGB 24-Bit to Amiga 3-Bit

```
//Match nearest color from the palette...
function rgb24_to_amiga3(p:pointer):byte;
var palette:pointer;maxid:byte;
begin
 palette := @LUT_AMIGA_3BIT[0];
 maxid   := 7;
```

Amiga 3-Bit to RGB 24-Bit

```
function amiga3_to_rgb24(amiga3:byte):TRgb;
begin
 if amiga3>7 then amiga3:=0;
 result:=LUT_AMIGA_3BIT[amiga3]; //a value of [0..7]
end;
```

Commodore Amiga 4-Bit

Commodore Amiga 4-Bit palette.
Total 16 different colors possible (2 pixels per byte).

ID	Color	Value	Name
0x00		000000	**Black**
0x01		561C1C	**Dark brown**
0x02		315600	**Dark green**
0x03		B70000	**Dark red**
0x04		68681C	**Dark yellow**
0x05		E31C1C	**Red**
0x06		686868	**Gray**
0x07		A85631	**Brown**
0x08		788998	**Blue**
0x09		78A831	**Light green**
0x0A		FF5631	**Light red**
0x0B		B78968	**Light brown**
0x0C		98A8B7	**Light blue**
0x0D		FF8956	**Orange**

```
const //Commodore Amiga 4-Bit lookup table (LUT)
 LUT_AMIGA_4BIT:array[0..$F] of TRgb=( //Amiga 4-Bit, 16 colors
  (b:$00;g:$00;r:$00), //00. #000000   AMIGA_4BIT_BLACK
  (b:$1C;g:$1C;r:$56), //01. #561C1C   AMIGA_4BIT_BROWN_DARK
  (b:$00;g:$56;r:$31), //02. #315600   AMIGA_4BIT_GREEN_DARK
  (b:$00;g:$00;r:$B7), //03. #B70000   AMIGA_4BIT_RED_DARK
  (b:$1C;g:$68;r:$68), //04. #68681C   AMIGA_4BIT_YELLOW_DARK
  (b:$1C;g:$1C;r:$E3), //05. #E31C1C   AMIGA_4BIT_RED
  (b:$68;g:$68;r:$68), //06. #686868   AMIGA_4BIT_GRAY
  (b:$31;g:$56;r:$A8), //07. #A85631   AMIGA_4BIT_BROWN
  (b:$98;g:$89;r:$78), //08. #788998   AMIGA_4BIT_BLUE
  (b:$31;g:$A8;r:$78), //09. #78A831   AMIGA_4BIT_GREEN_LIGHT
  (b:$31;g:$56;r:$FF), //10. #FF5631   AMIGA_4BIT_RED_LIGHT
  (b:$68;g:$89;r:$B7), //11. #B78968   AMIGA_4BIT_BROWN_LIGHT
  (b:$B7;g:$A8;r:$98), //12. #98A8B7   AMIGA_4BIT_BLUE_LIGHT
  (b:$56;g:$89;r:$FF), //13. #FF8956   AMIGA_4BIT_ORANGE
  (b:$78;g:$C6;r:$FF), //14. #FFC678   AMIGA_4BIT_YELLOW
  (b:$D4;g:$E3;r:$F1)  //15. #F1E3D4   AMIGA_4BIT_WHITE
 );
```

RGB 24-Bit to Amiga 4-Bit

```
//Match nearest color from the palette...
function rgb24_to_amiga4(p:pointer):byte;
var palette:pointer;maxid:byte;
begin
 palette := @LUT_AMIGA_4BIT[0];
 maxid   := $F;
 result  := nearestLutColor(palette,p,maxid); //see above
end;
```

Amiga 4-Bit to RGB 24-Bit

```
function amiga4_to_rgb24(amiga4:byte):TRgb;
begin
 if amiga4>15 then amiga4:=0;
 result:=LUT_AMIGA_4BIT[amiga4]; //a value of [0..15]
end;
```

ID	Color	Value	Name	ID	Color	Value	Name
0x00		000000	Black	0x10		779999	Dark cyan
0x01		331100	Dark brown	0x11		FF6644	Red lighter
0x02		112200	Darker green	0x12		CC8866	Orange dark
0x03		990000	Darker red	0x13		8899DD	Blue light
0x04		333333	Darker gray	0x14		99AAAA	Gray
0x05		663311	Brown	0x15		88BB77	Light green 2
0x06		335500	Dark green	0x16		FF9944	Orange
0x07		DD1111	Dark red	0x17		AACC33	Lime green
0x08		665555	Dark gray	0x18		BBBBBB	Light gray
0x09		AA3322	Red	0x19		FFBB55	Dark yellow
0x0A		448811	Green	0x1A		FFAA88	Light orange
0x0B		666688	Blue	0x1B		BBDDEE	Light cyan
0x0C		AA6644	Light brown	0x1C		EECCCC	Light purple
0x0D		FF4422	Light red	0x1D		FFDD66	Yellow
0x0E		997788	Purple	0x1E		FFFF99	Light yellow
0x0F		779933	Light green	0x1F		FFEEDD	White

```
const //Commodore Amiga 5-Bit lookup table (LUT)
 LUT_AMIGA_5BIT:array[0..31] of TRgb=( //Amiga 5-Bit, 32 colors
  (b:$00;g:$00;r:$00), //00. #000000   AMIGA_5BIT_BLACK
  (b:$00;g:$11;r:$33), //01. #331100   AMIGA_5BIT_BROWN_DARK
  (b:$00;g:$22;r:$11), //02. #112200   AMIGA_5BIT_GREEN_DARKER
  (b:$00;g:$00;r:$99), //03. #990000   AMIGA_5BIT_RED_DARKER
  (b:$33;g:$33;r:$33), //04. #333333   AMIGA_5BIT_GRAY_DARKER
```

```
  (b:$22;g:$33;r:$AA), //09. #AA3322   AMIGA_5BIT_RED
  (b:$11;g:$88;r:$44), //10. #448811   AMIGA_5BIT_GREEN
  (b:$88;g:$66;r:$66), //11. #666688   AMIGA_5BIT_BLUE
  (b:$44;g:$66;r:$AA), //12. #AA6644   AMIGA_5BIT_BROWN_LIGHT
  (b:$22;g:$44;r:$FF), //13. #FF4422   AMIGA_5BIT_RED_LIGHT
  (b:$88;g:$77;r:$99), //14. #997788   AMIGA_5BIT_PURPLE
  (b:$33;g:$99;r:$77), //15. #779933   AMIGA_5BIT_GREEN_LIGHT
  (b:$99;g:$99;r:$77), //16. #779999   AMIGA_5BIT_CYAN_DARK
  (b:$44;g:$66;r:$FF), //17. #FF6644   AMIGA_5BIT_RED_LIGHTER
  (b:$66;g:$88;r:$CC), //18. #CC8866   AMIGA_5BIT_ORANGE_DARK
  (b:$DD;g:$99;r:$88), //19. #8899DD   AMIGA_5BIT_BLUE_LIGHT
  (b:$AA;g:$AA;r:$99), //20. #99AAAA   AMIGA_5BIT_GRAY
  (b:$77;g:$BB;r:$88), //21. #88BB77   AMIGA_5BIT_GREEN_LIGHT2
  (b:$44;g:$99;r:$FF), //22. #FF9944   AMIGA_5BIT_ORANGE
  (b:$33;g:$CC;r:$AA), //23. #AACC33   AMIGA_5BIT_GREEN_LIME
  (b:$BB;g:$BB;r:$BB), //24. #BBBBBB   AMIGA_5BIT_GRAY_LIGHT
  (b:$55;g:$BB;r:$FF), //25. #FFBB55   AMIGA_5BIT_YELLOW_DARK
  (b:$88;g:$AA;r:$FF), //26. #FFAA88   AMIGA_5BIT_ORANGE_LIGHT
  (b:$EE;g:$DD;r:$BB), //27. #BBDDEE   AMIGA_5BIT_CYAN_LIGHT
  (b:$CC;g:$CC;r:$EE), //28. #EECCCC   AMIGA_5BIT_GRAY_LIGHTER
  (b:$66;g:$DD;r:$FF), //29. #FFDD66   AMIGA_5BIT_YELLOW
  (b:$99;g:$FF;r:$FF), //30. #FFFF99   AMIGA_5BIT_YELLOW_LIGHT
  (b:$DD;g:$EE;r:$FF)  //31. #FFEEDD   AMIGA_5BIT_WHITE
 );
```

RGB 24-Bit to Amiga 5-Bit

```
//Match nearest color from the palette...
function rgb24_to_amiga5(p:pointer):byte;
var palette:pointer;maxid:byte;
begin
 palette := @LUT_AMIGA_5BIT[0];
 maxid   := $1F;
 result  := nearestLutColor(palette,p,maxid); //see above
end;
```

Amiga 5-Bit to RGB 24-Bit

```
end;
```

ZX Spectrum 4-Bit
ZX Spectrum palette, 4-Bit.
Total 16 colors possible (2 pixels per byte).
RGBi variation (15 colors and a duplicate black).

ID	Color	Value	Name
0x00		000000	**Black**
0x01		0000AA	**Dark blue**
0x02		AA0000	**Dark red**
0x03		AA00AA	**Dark magenta**
0x04		00AA00	**Dark green**
0x05		00AAAA	**Dark cyan**
0x06		AAAA00	**Dark yellow**
0x07		AAAAAA	**Gray**
0x08		000000	**Duplicate black**
0x09		0000FF	**Blue**
0x0A		FF0000	**Red**
0x0B		FF00FF	**Magenta**
0x0C		00FF00	**Green**
0x0D		00FFFF	**Cyan**
0x0E		FFFF00	**Yellow**
0x0F		FFFFFF	**White**

```
  (b:$AA;g:$00;r:$00), //01. #0000AA   SPECTRUM_BLUE_DARK
  (b:$00;g:$00;r:$AA), //02. #AA0000   SPECTRUM_RED_DARK
  (b:$AA;g:$00;r:$AA), //03. #AA00AA   SPECTRUM_MAGENTA_DARK
  (b:$00;g:$AA;r:$00), //04. #00AA00   SPECTRUM_GREEN_DARK
  (b:$AA;g:$AA;r:$00), //05. #00AAAA   SPECTRUM_CYAN_DARK
  (b:$00;g:$AA;r:$AA), //06. #AAAA00   SPECTRUM_YELLOW_DARK
  (b:$AA;g:$AA;r:$AA), //07. #AAAAAA   SPECTRUM_GRAY
  (b:$00;g:$00;r:$00), //08. #000000   SPECTRUM_BLACK_DUPL
  (b:$FF;g:$00;r:$00), //09. #0000FF   SPECTRUM_BLUE
  (b:$00;g:$00;r:$FF), //10. #FF0000   SPECTRUM_RED
  (b:$FF;g:$00;r:$FF), //11. #FF00FF   SPECTRUM_MAGENTA
  (b:$00;g:$FF;r:$00), //12. #00FF00   SPECTRUM_GREEN
  (b:$FF;g:$FF;r:$00), //13. #00FFFF   SPECTRUM_CYAN
  (b:$00;g:$FF;r:$FF), //14. #FFFF00   SPECTRUM_YELLOW
  (b:$FF;g:$FF;r:$FF)  //15. #FFFFFF   SPECTRUM_WHITE
 );
```

RGB 24-Bit to ZX Spectrum 4-Bit

```
const
 RGBI_TO_SPECTRUM:array[0..$F] of byte=(0,1,4,5,2,3,6,7,8,9,12,13,10,11,14,15);

function rgb24_to_spectrum(p:pointer):byte;
var rgbi:byte;
begin
 rgbi:=rgb24_to_rgbi(p); //first convert to RGBi (4-Bit)
 result:=RGBI_TO_SPECTRUM[rgbi]; //then pick related color from array
end;
```

Amstrad CPC 5-Bit

Amstrad CPC palette, 5-Bit.
Total 27 colors used.

ID	Color	Value	Name	ID	Color	Value	Name
0x00		000000	**Black**	0x10		FF8080	**Pink**
0x01		000080	**Blue**	0x11		FF80FF	**Pale magenta**
0x02		0000FF	**Bright blue**	0x12		00FF00	**Bright green**
0x03		800000	**Red**	0x13		00FF80	**Sea green**
0x04		800080	**Magenta**	0x14		00FFFF	**Bright cyan**
0x05		8000FF	**Violet**	0x15		80FF00	**Lime green**
0x06		FF0000	**Bright red**	0x16		80FF80	**Pale green**
0x07		FF0080	**Purple**	0x17		80FFFF	**Pale cyan**
0x08		FF00FF	**Bright magenta**	0x18		FFFF00	**Bright yellow**
0x09		008000	**Green**	0x19		FFFF80	**Pale yellow**
0x0A		008080	**Cyan**	0x1A		FFFFFF	**White**
0x0B		0080FF	**Sky blue**				
0x0C		808000	**Yellow**				
0x0D		808080	**Gray**				
0x0E		8080FF	**Pale blue**				
0x0F		FF8000	**Orange**				

```
const //Amstrad CPC 5-Bit lookup table (LUT)
 LUT_AMSTRAD_5BIT:array[0..26] of TRgb=( //Amstrad CPC
  (b:$00;g:$00;r:$00), //00. #000000   AMSTRAD_BLACK
  (b:$80;g:$00;r:$00), //01. #000080   AMSTRAD_BLUE
  (b:$FF;g:$00;r:$00), //02. #0000FF   AMSTRAD_BLUE_BRIGHT
  (b:$00;g:$00;r:$80), //03. #800000   AMSTRAD_RED
  (b:$80;g:$00;r:$80), //04. #800080   AMSTRAD_MAGENTA
  (b:$FF;g:$00;r:$80), //05. #8000FF   AMSTRAD_VIOLET
  (b:$00;g:$00;r:$FF), //06. #FF0000   AMSTRAD_RED_BRIGHT
  (b:$80;g:$00;r:$FF), //07. #FF0080   AMSTRAD_PURPLE
  (b:$FF;g:$00;r:$FF), //08. #FF00FF   AMSTRAD_MAGENTA_BRIGHT
  (b:$00;g:$80;r:$00), //09. #008000   AMSTRAD_GREEN
```

```
  (b:$FF;g:$80;r:$80), //14. #8080FF   AMSTRAD_BLUE_PALE
  (b:$00;g:$80;r:$FF), //15. #FF8000   AMSTRAD_ORANGE
  (b:$80;g:$80;r:$FF), //16. #FF8080   AMSTRAD_PINK
  (b:$FF;g:$80;r:$FF), //17. #FF80FF   AMSTRAD_MAGENTA_PALE
  (b:$00;g:$FF;r:$00), //18. #00FF00   AMSTRAD_GREEN_BRIGHT
  (b:$80;g:$FF;r:$00), //19. #00FF80   AMSTRAD_GREEN_SEA
  (b:$FF;g:$FF;r:$00), //20. #00FFFF   AMSTRAD_CYAN_BRIGHT
  (b:$00;g:$FF;r:$80), //21. #80FF00   AMSTRAD_GREEN_LIME
  (b:$80;g:$FF;r:$80), //22. #80FF80   AMSTRAD_GREEN_PALE
  (b:$FF;g:$FF;r:$80), //23. #80FFFF   AMSTRAD_CYAN_PALE
  (b:$00;g:$FF;r:$FF), //24. #FFFF00   AMSTRAD_YELLOW_BRIGHT
  (b:$80;g:$FF;r:$FF), //25. #FFFF80   AMSTRAD_YELLOW_PALE
  (b:$FF;g:$FF;r:$FF)  //26. #FFFFFF   AMSTRAD_WHITE
 );
```

RGB 24-Bit to Amstrad CPC 5-Bit

```
//Match nearest color from the palette...
function rgb24_to_amstrad(p:pointer):byte;
var palette:pointer;maxid:byte;
begin
 palette := @LUT_AMSTRAD_5BIT[0];
 maxid   := 26;
 result  := nearestLutColor(palette,p,maxid); //see above
end;
```

Amstrad CPC 5-Bit to RGB 24-Bit

```
function amstrad5_to_rgb24(n:byte):TRgb;
begin
 if n>26 then n:=0;
 result:=LUT_AMSTRAD_5BIT[n]; //a value of [0..26]
end;
```

Nintendo GameBoy 2-Bit

Nintendo GameBoy palette, 2-Bit.
Total 4 shades of green possible (4 pixels per byte).

ID	Color	Value	Name
0x00		0F380F	**Darkest**
0x01		306230	**Dark**
0x02		7DA114	**Bright**
0x03		9BBC0F	**Brightest**

```
const //Nintendo GameBoy 2-Bit lookup table (LUT)
 LUT_GAMEBOY_2BIT:array[0..3] of TRgb=( //GameBoy 2-Bit
  (b:$0F;g:$38;r:$0F), //00. #0F380F   GAMEBOY_DARKEST
  (b:$30;g:$62;r:$30), //01. #306230   GAMEBOY_DARK
  (b:$14;g:$A1;r:$7D), //02. #7DA114   GAMEBOY_BRIGHT
  (b:$0F;g:$BC;r:$9B)  //03. #9BBC0F   GAMEBOY_BRIGHTEST
 );
```

RGB 24-Bit to GameBoy 2-Bit

```
function rgb24_to_gameboy(p:pointer):byte; //same thing as: rgb24_to_gray2
var gray8:byte;
begin
 with pRgb(p)^ do gray8 := (r+g+b) div 3; //first convert to gray8
 result := gray8 shr 6; //shr 6: get rid of 6 bits
end;
```

GameBoy 2-Bit to RGB 24-Bit

```
function gameboy_to_rgb24(n:byte):TRgb;
begin
 if n>3 then n:=0;
```

Nintendo NES 6-Bit

Nintendo Entertainment System palette, 6-Bit.
Total 64 colors of which 8 colors are unused.

ID	Color	Value	Name	ID	Color	Value	Name
0x00		7C7C7C	Gray	0x20		F8F8F8	Lightest gray
0x01		0000FC	Blue	0x21		3CBCFC	Sky blue
0x02		0000BC	Dark blue	0x22		6888FC	Medium blue
0x03		4428BC	Viola	0x23		9878F8	Lighter viola
0x04		940084	Dark magenta	0x24		F878F8	Light magenta
0x05		A80020	Dark red	0x25		F85898	Light pink
0x06		A81000	Dark orange	0x26		F87858	Light red
0x07		881400	Darker red	0x27		FCA044	Light orange
0x08		503000	Brown	0x28		F8B800	Yellow
0x09		007800	Medium green	0x29		B8F818	Lime green
0x0A		006800	Dark green	0x2A		58D854	Light green
0x0B		005800	Darker green	0x2B		58F898	Turquoise
0x0C		004058	Darkest cyan	0x2C		00E8D8	Dark cyan
0x0D		000000	Black	0x2D		787878	Darker gray
0x0E		-	Unused 1	0x2E		-	Unused 5
0x0F		-	Unused 2	0x2F		-	Unused 6
0x10		BCBCBC	Gray	0x30		FCFCFC	White
0x11		0078F8	Light blue	0x31		A4E4FC	Light cyan
0x12		0058F8	Medium blue	0x32		B8B8F8	Lighter blue
0x13		6844FC	Light viola	0x33		D8B8F8	Light purple
0x14		D800CC	Magenta	0x34		F8B8F8	Lighter magenta
0x15		E40058	Pink	0x35		F8A4C0	Lighter pink

0x1A	00A800	**Medium green 1**	0x3A	B8F8B8	**Lighter green**
0x1B	00A844	**Medium green 2**	0x3B	B8F8D8	**Light turquoise**
0x1C	008888	**Darker cyan**	0x3C	00FCFC	**Cyan**
0x1D	000000	**Duplicate black**	0x3D	D8D8D8	**Light gray**
0x1E	-	**Unused 3**	0x3E	-	**Unused 7**
0x1F	-	**Unused 4**	0x3F	-	**Unused 8**

NES 6-Bit to RGB 24-Bit

```
function nes_to_rgb24(nes6:byte):TRgb;
begin
 if nes6>63 then nes6:=0;
 result:=LUT_NES_6BIT[nes6]; //a value of [0..63]
end;
```

Related pages of Colors and formats

- Programming
- Commodore 64
- Bytes and data-types
- Delphi programming
- File formats and fileinfo