```
#define BIT0 1
#define BIT1 (1<<1)
#define BIT2 (1<<2)
...
#define BIT31 (1<<31)
```

```
k |= (BIT6 | BIT7);
```

```
k &= ~(BIT0 | BIT3);
```

```
k |= (BIT1|BIT3|BIT5|BIT7);
k &= ~(BIT0|BIT2|BIT4|BIT6);
```

```
k ^= BIT0;
k |= BIT5;
k &= ~BIT7;
```

```
#include <stdbool.h>
...
bool res = ((k & BIT4) && !(k & BIT6));
```

$$Light\ 0:000 \quad Green:001 \quad High:11 \implies 0x07$$

$$0xAB \implies Light\ 5:101 \quad Red:010 \quad High:11$$

```
//Assuming Standard C.
//BITx macro equivalents are available in the comments.
#define INTENSITY_MASK (0x03) // (BIT0 | BIT1)
#define COLOR_MASK (0x1C) // (BIT2 | BIT3 | BIT4)
#define LAMP_MASK (0xE0) // (BIT5 | BIT6 | BIT7)
```

```
//Using Octal Codes.
typedef enum INTENSITY {
  OFF = (00),
  LOW = (01),
  MEDIUM = (02),
  HIGH = (03)
} INTENSITY;

typedef enum COLOR {
  BLUE = (00 << 2),
  GREEN = (01 << 2),
  RED = (02 << 2),
  YELLOW = (03 << 2),
  WHITE = (04 << 2)
} COLOR;
```

```
// Example:extractColor(0xC3) returns BLUE
COLOR extractColor(int LSC)
{
    return (LSC & COLOR_MASK);
}
```

```
// Example:makeLSC(6, BLUE, HIGH_INTENSITY) returns 0xC3
int makeLSC(int lampNumber, COLOR newColor, INTENSITY
   newIntensity)
{
  return (((lampNumber<<5) & LAMP_MASK)
            | newIntensity
            | newColor);
}
```

```
for (uint8_t i = 0; i < 8; ++i) {
  if (currentIntensity[i] != HIGH) ++currentIntensity[i];
  LSC = makeLSC(i, currentColor[i], currentIntensity[i]);
}
```

```
LSC = 0xC3; // This sends the "turn lamp #6 to high "blue command
LSC = 0x11; // This sends the "turn lamp #0 to low "white command
```