Q: How do we blink an LED while keeping the other ON at all times.

Ans: We cannot set the blue LED on along with the red LED, because data is being written to the same register (GPIO_DATAF). This is where we can utilize bitwise operators.

From Code

The defined variable c shows bitwise results of A, B:

Locals

Variable	Value	Location
a	0b101'1010'0101'1010'0101'1010'0101'...	0x2000'03f4
b	0b1101'1110'1010'1101'1011'1110'1110...	0x2000'03ec
c	0b1101'1110'1111'1111'1111'1110'1111...	0x2000'03f0
count	<unavailable>	

a OR b

Locals

Variable	Value	Location
a	0b101'1010'0101'1010'0101'1010'0101'...	0x2000'03f4
b	0b1101'1110'1010'1101'1011'1110'1110...	0x2000'03ec
c	0b101'1010'0000'1000'0001'1010'0100'...	0x2000'03f0
count	<unavailable>	

a AND B

Locals

Variable	Value	Location
a	0b101'1010'0101'1010'0101'1010'0101'1010	0x2000'03f4
b	0b1101'1110'1010'1101'1011'1110'1110'1111	0x2000'03ec
c	0b1000'0100'1111'0111'1110'0100'1011'0101	0x2000'03f0
count	<unavailable>	

a XOR b

The operations take only 1 machine instruction for XOR, AND, OR and NOT ⟶

```
unsigned int b = 0xDEADBEEF;
        0x46: 0x491c          LDR.N     R1, ??mai
c = a | b;     // a OR b
        0x48: 0xea51 0x0300   ORRS.W    R3, R1, R
c = a & b;     // a AND b
        0x4c: 0xea11 0x0400   ANDS.W    R4, R1, R
c = a ^ b;     // a XOR b
        0x50: 0xea91 0x0500   EORS.W    R5, R1, R
c = ~b;        // b NOT
        0x54: 0x43ce          MVNS      R6, R1
```

The rightshift operand is: c = b >> 1; and it shifts all the bits of b to the right by 1.

LSRS shifts 0 to the most significant bit position in b

```
c = b >> 1;    // b Left shift
        0x56: 0x000f          MOVS      R7, R1
        0x58: 0x087f          LSRS      R7, R7, #1
```

This is equivalent to division by 2^1

Variable	Value
a	0b101'1010'0101'1010'0101'1010'0101'1010
b	3'735'928'559
c	1'867'964'279
count	<unavailable>

This is similar for the left shift: c=b<<3; shifts by 3

Variable	Value
a	0b101'1010'0101'1010'0101'1010'0101'1010
b	0b1101'1110'1010'1101'1011'1110'1110'1111
c	0b1111'0101'0110'1101'1111'0111'0111'1000
count	<unavailable>

bits shifted right by 3 positions

This is equivalent to multiplying b by 2^3. Zeros are shifted to Least significant bit positions.

Shifting with signed bits

If bits are signed, ex — x=1024, y=-1024 then right shifting will introduce a 0 or 1 depending on number being shifted.

For z=x>>3;

x	1'024	R3 int
y	-1'024	R4 int
z	128	R10 int
count	<unavailable>	

Here 0's get shifted to the most significant bit because shift operation is on +ve number

x	1'024	R3 int
y	-1'024	R4 int
z	-128	R5 int
count	<unavailable>	

Errors 0, Warnings 0 Ln 28, Col

Here 1's get shifted to the most significant bit position because shift operation is on -ve number. The machine code in the dissassembly is ASRS or arithmatic right shift (vs LSRs or logical right shift for unsigned numbers).

```
0x68: 0x469a          MOV     R10, R3
0x6a: 0xea5f 0x0aea   ASRS.W  R10, R10, #3
z = y >> 3;
0x6e: 0x0025          MOVS    R5, R4
0x70: 0x10ed          ASRS    R5, R5, #3
SYSCTL_RCGCGPIO_R = 0x20u;      //u to indicate unsig
0x72: 0xf05f 0x0c20   MOVS.W  R12, #32
```

To write data to GPIO register, the program first reads data from GPIO-register using LDR.N R0, [PC, #0x3C] and stores it in register R0. Since we are using bit masking of bits 9:2 of GPIO-data F register, the data from this register is 0X400253FC (corresponds to bits 9:2 set by default). This