CS/COE 0447

Bitfields and Masking

wilkie (with content borrowed from:
Jarrett Billingsley
Dr. Bruce Childers)

Bitfields

If you build bits, they will come.

Remember bit *flags?*

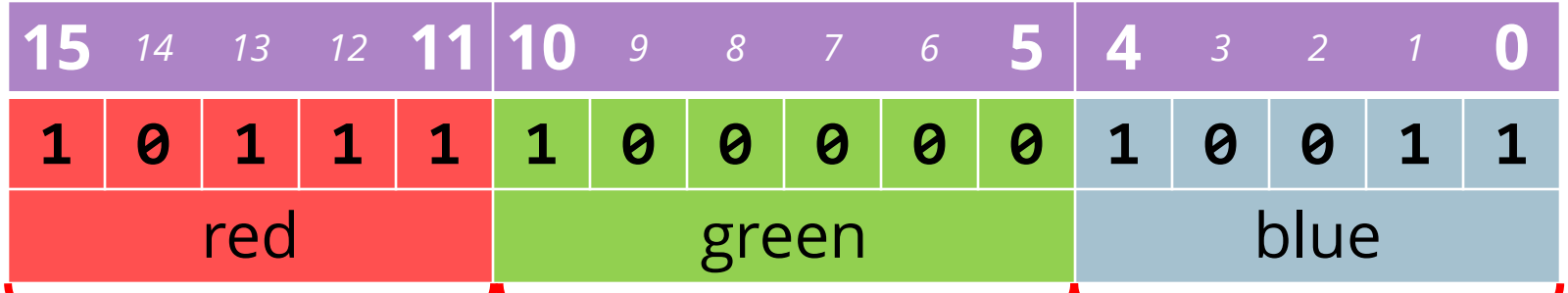
- this is when you treat a pattern of bits as a set of **booleans**

0 0 0 0 0 0 0 0 0 0 0 0 0

B R L D U

The masters of meaning

- well what if we wanted to store multiple ***integers*** in one value?



decimal?

23 **32** **19**

That's this color, in RGB565.

Why Do This To Ourselves???

- It's **smaller**
  - Really, that's it.
  - But that's super important in a lot of cases.
- **Smaller data...**
  - Takes up **less space** in memory.
  - Takes up **less space** in *cache.*
    - Extremely important thing in modern CPUs that we talk about in 1541.
  - Is **faster to move** between memory and the CPU.
  - Is **faster to transfer** across the internet and other networks.

I wanna turn the light on!!

- I have a sequence of 0s. I wanna turn one of them into a 1.
- What **bitwise operation** can I use to do that?

```
  0  0  0  0
? 1  0  0  0
-------------
  1  0  0  0
```

I wanna turn the light off!!

- I wanna turn one of the 1s into a 0.
- What **bitwise operation** can I use to do *that?*

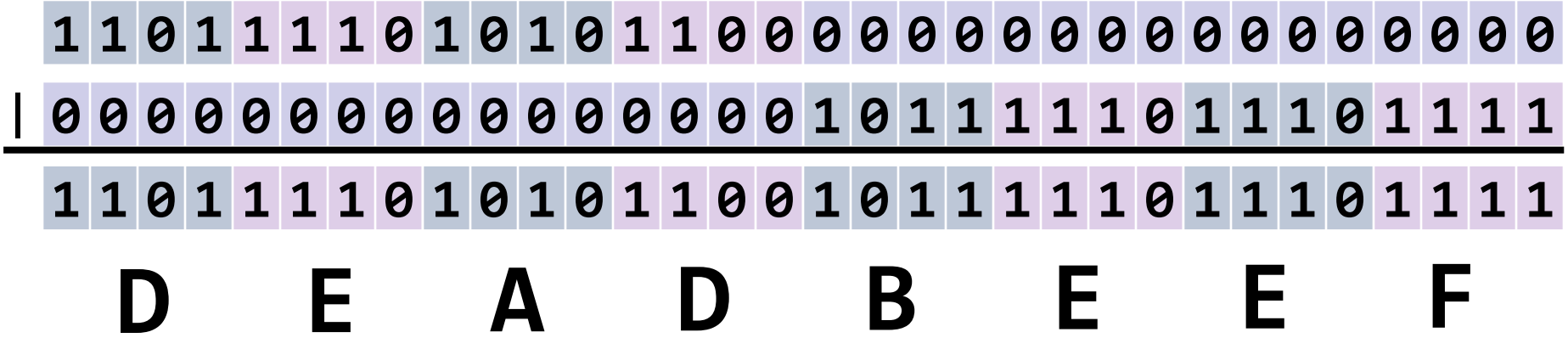
Turning off the first three, leaving the others alone

- More bits, but this is one of the same operations…

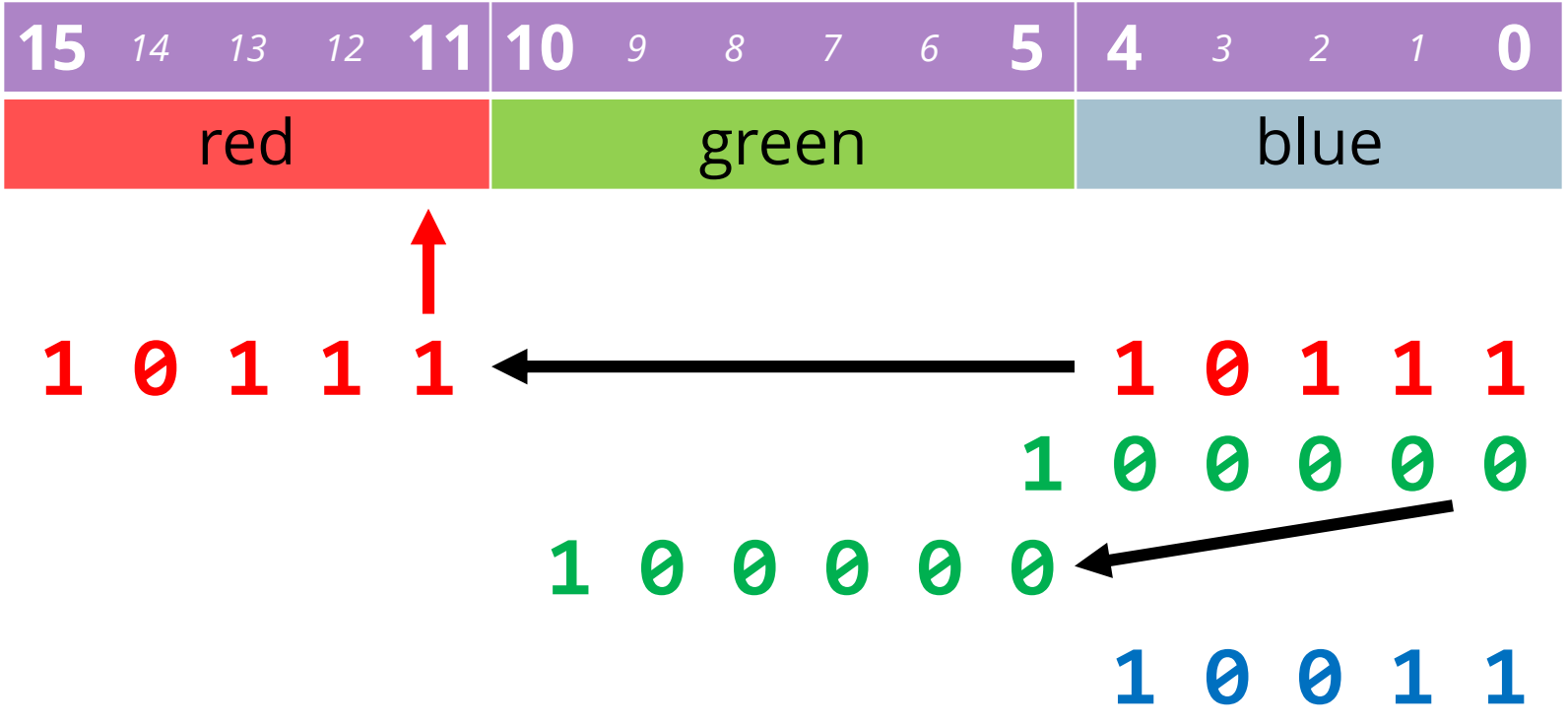
```
   1   0   1   0   1   1
?  0   0   0   1   1   1
-------------------------
   0   0   0   0   1   1
```

Remember this?

```
lui at, 0xDEAD
ori t0, at, 0xBEEF
```



Let's look at this again.



...

- hmm

```
1 0 1 1 1   0 0 0 0 0 0 0 0 0 0 0

0 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 1 0 0 1 1
-----------------------------------
1 0 1 1 1   1 0 0 0 0 0 1 0 0 1 1
```

Left-shifting and ORing

- If you have the **values of the fields**.
- And you want to **put them together into a bitfield**.
  - **Shift** each value **left** to the correct bit position.
  - **OR** the **shifted values** together.
- For RGB565,
  - Red is shifted left 11
  - Green is shifted left 5
  - Blue isn't shifted (shifted left 0...)

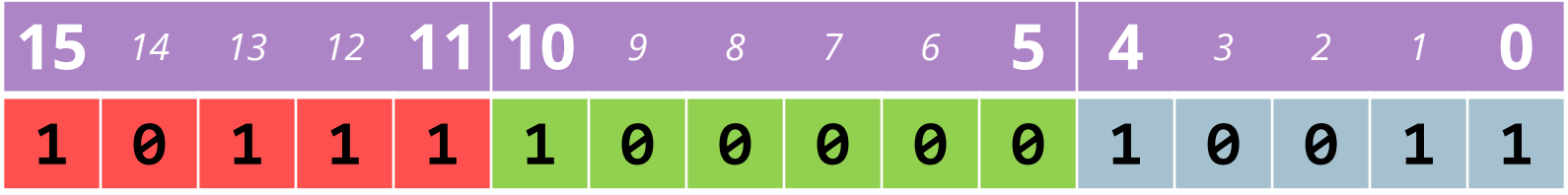
```
color = (red << 11) | (green << 5) | blue;
```

Masking

Getting rid of the bits that wronged us.

Going the other way

- Let's go from the bitfield to **three separate values.**



Let's say we somehow set all the non-red bits to 0.

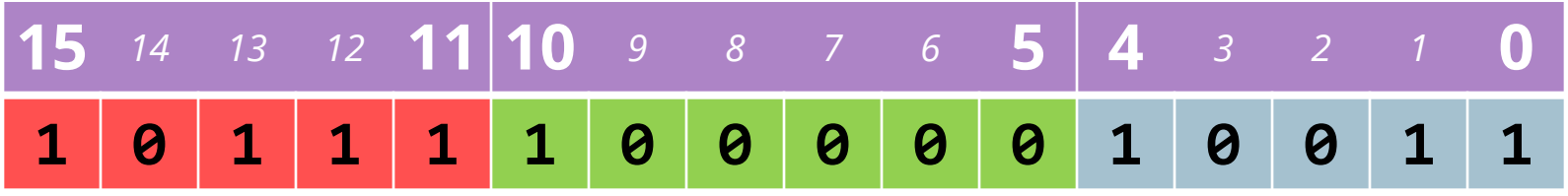
What value is this?

It's not 23, that's for sure.

So **how do we fix that?**

It's the exact opposite

- We have to **shift right** to put the field at position 0.



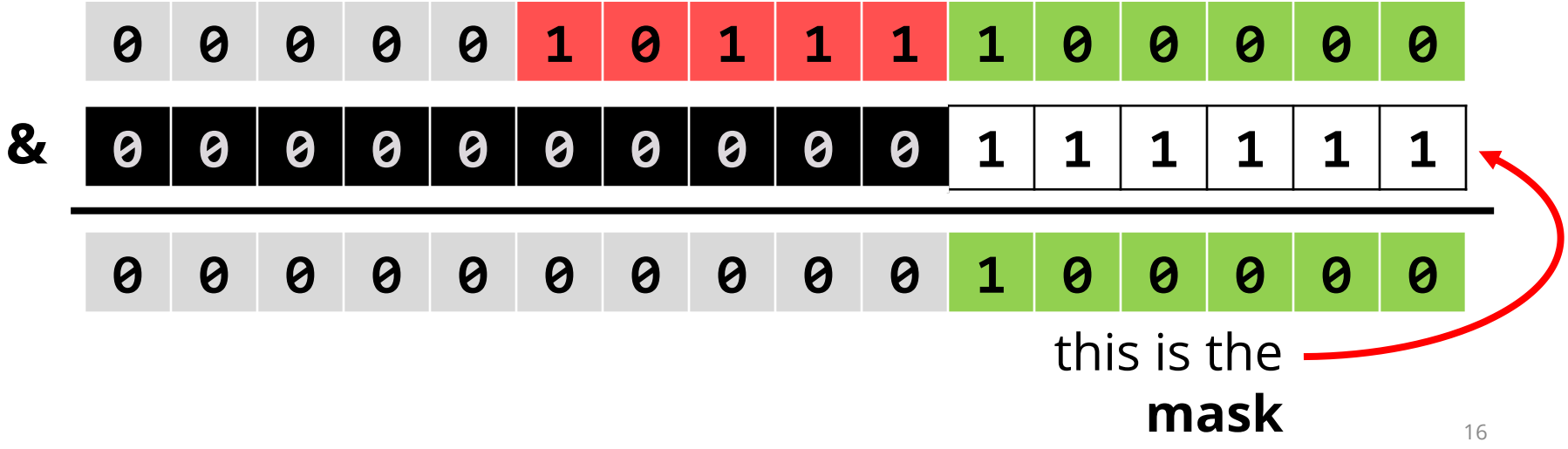
shift right by 11 and...

cool. what about green? shift right by...?

Uh oh.

Masquerade

- We need to **get rid of (zero out)** the bits that we don't care about.
- A **mask** is a **specially-constructed value** that has:
  - 1s in the bits that we want to keep
  - 0s in the bits that we want to discard
- Which bits do we want to keep? which do we want to discard? (aka, light bulbs we want to turn off)



Coming up with the mask value

- If you want to mask a 3 bit value, the mask is $\mathbf{111_2}$
- If you want to mask a 4 bit value, the mask is $\mathbf{1111_2}$
- If you want to mask a 5 bit value, it's...?

Size(n)	Mask	2^n	Mask in decimal
3	111_2	8	7
4	1111_2	16	15
5	11111_2	32	31

Right-shifting and ANDing

- To **extract** one or more fields from a bitfield:
  - **Shift** the bitfield **right** to put the desired field at bit position 0
  - **AND** that with **2^n-1**, where *n* is the number of bits in the field

- So for RGB565…
  - The red and blue masks are 2^5-1 = 31 (or **0x1F**)
  - The green mask is 2^6-1 = 63 (or **0x3F**)

```
red   = (color >> 11) & 0x1F;
green = (color >> 5)  & 0x3F;
blue  =  color        & 0x1F;
```

NOW it works

- Let's extract green:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	0	0	0	0	0	1	0	0	1	1

shift right by 5 and...

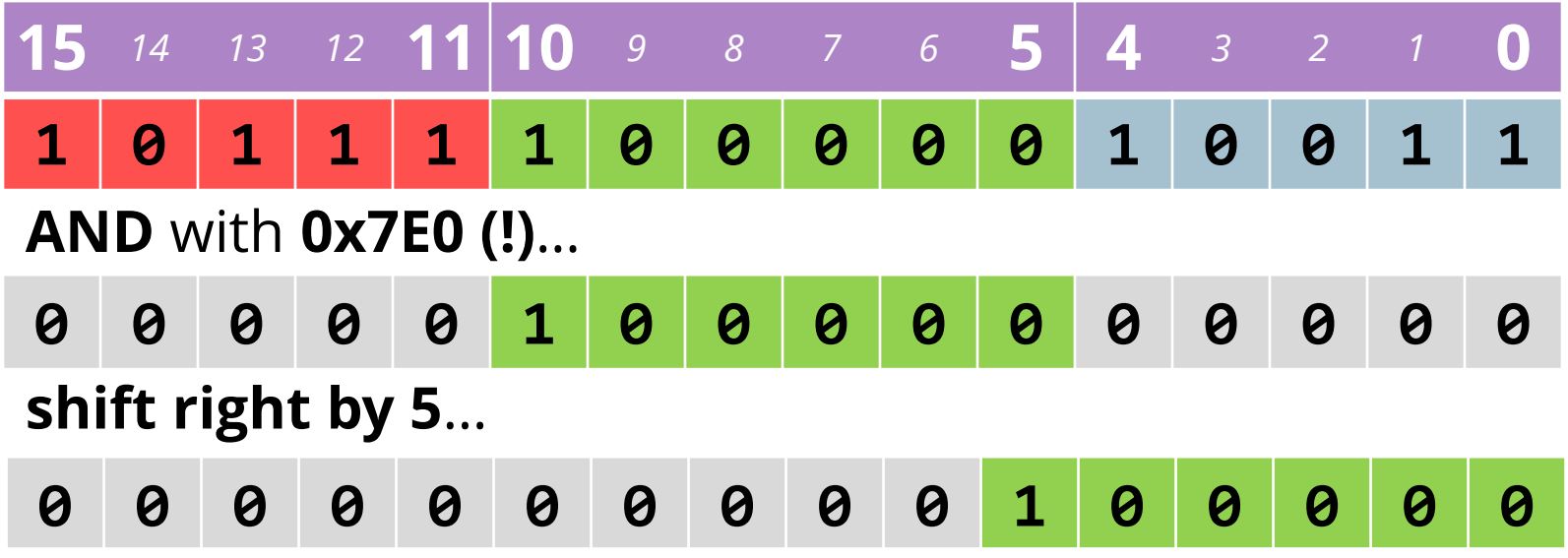
0	0	0	0	0	1	0	1	1	1	1	0	0	0	0	0

and then **AND** with **0x3F**...

0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

Can't you AND *then* shift?

- Sure, but...



where did I get 0x7E0??

it's 0x3F << 5. I feel like that's uglier.