Hex Formula Calculation

Table of Contents

- Idea: add them?
- Idea: xor?
- Idea: just c shifted?
- Idea: b shifted?
- Another idea:

formulate a question for wolfram alpha from this:

i have the following table with hex values. tell me a formula which can calculate the out column of all lines using the b and c columns.

b	c	out
0B	8B	05d000
08	C0	047000
02	6E	014000
01	DC	00f000
06	02	031000

Sure — here's a Wolfram Alpha–friendly phrasing of your question:

Given the table below with hexadecimal values, find a formula that computes the `out` *column from columns* `b` *and* `c`*.*

b c out

0B 8B 05D000

08 C0 047000

02 6E 014000

01 DC 00F000

06 02 031000

```css
> Please express the formula in terms of `b` and `c` that matches all rows.

If you want, I can also help analyze this myself — just say **"yes"** and I'
```

yes

OK — let's break this down carefully. First, let's restate the data clearly in decimal for easier reasoning:

b (hex)	c (hex)	out (hex)	b (dec)	c (dec)	out (dec)
0B	8B	05D000	11	139	380,928
08	C0	047000	8	192	290,816
02	6E	014000	2	110	81,920
01	DC	00F000	1	220	61,440
06	02	031000	6	2	200,704

Let's notice that all `out` values are multiples of 4096 (0x1000), since: $0x1000 = 4096$

So if we factor:

- 05D000 / 4096 = 0x5D000 / 0x1000 = 0x5D = 93 decimal
- 047000 / 4096 = 0x47 = 71
- 014000 / 4096 = 0x14 = 20
- 00F000 / 4096 = 0xF = 15
- 031000 / 4096 = 0x31 = 49

So effectively the table of interest is:

b	c	out / 4096
11	139	93
8	192	71
2	110	20
1	220	15
6	2	49

Our job is to find a rule that maps (b,c) to this "out divided by 4096" column.

Let's look for a pattern. Let's try to see if `out/4096` is proportional to b and c somehow.

Quick differences:

b	c	out/4096
11	139	93
8	192	71
2	110	20
1	220	15
6	2	49

No obvious linear trend. Let's test a linear model:
$k_1 \times b + k_2 \times c + k_3 \approx \text{out}/4096$

Let's set up some examples:

1. $11k_1 + 139k_2 + k_3 = 93$
2. $8k_1 + 192k_2 + k_3 = 71$
3. $2k_1 + 110k_2 + k_3 = 20$

This is a solvable system with 3 equations, 3 unknowns. I can solve it step by step if you'd like.

Or, if you'd rather I try to infer a *pattern* (bit shifts, masks, etc.), let me know — it might be more meaningful if this is firmware-like data.

👉 Which approach do you want?

✅ Solve the linear system
✅ Try to spot a hardware-style pattern (shifts/masks)

✅ Something else

Try to spot a hardware-style pattern

OK — let's think like an embedded engineer: patterns with shifts, masks, concatenations, or lookups.

First, note:

- `out` values are all multiples of `0x1000` → lowest 12 bits zero
  - So the significant part is the upper bits
- That means the "interesting" part is `out >> 12`:

b	c	out >> 12 (hex)
0B	8B	0x5D (93)
08	C0	0x47 (71)
02	6E	0x14 (20)
01	DC	0x0F (15)
06	02	0x31 (49)

Let's work in hex, easier for hardware:

b	c	out_shifted
0B	8B	5D
08	C0	47
02	6E	14
01	DC	0F
06	02	31

Maybe some combination of `b` and `c` bits produces these? Let's try:

Idea: add them?

```r
```

```
b + c
0x0B + 0x8B = 0x96
0x08 + 0xC0 = 0xC8
0x02 + 0x6E = 0x70
0x01 + 0xDC = 0xDD
0x06 + 0x02 = 0x08
```

No match to `out`.

Idea: xor?

```nginx
b ^ c
0x0B ^ 0x8B = 0x80
0x08 ^ 0xC0 = 0xC8
0x02 ^ 0x6E = 0x6C
0x01 ^ 0xDC = 0xDD
0x06 ^ 0x02 = 0x04
```

Nope.

Idea: just c shifted?

```r
c >> 1
8B >> 1 = 0x45
C0 >> 1 = 0x60
6E >> 1 = 0x37
DC >> 1 = 0x6E
02 >> 1 = 0x01
```

Nope.

Idea: b shifted?

```bash
b << 3
0B << 3 = 0x58
08 << 3 = 0x40
02 << 3 = 0x10
01 << 3 = 0x08
06 << 3 = 0x30
```

Reasonably close to:

```r
58 vs 5D
40 vs 47
10 vs 14
08 vs 0F
30 vs 31
```

Aha! That's promising.

Let's see the difference:

```r
5D - 58 = 5
47 - 40 = 7
14 - 10 = 4
0F - 08 = 7
31 - 30 = 1
```

So roughly:

```ini
out_shifted = (b << 3) + offset
```

where offset is varying. Could the offset come from c? Let's look:

```r
c & 0x07   (lowest 3 bits)
8B & 7 = 3
C0 & 7 = 0
6E & 7 = 6
DC & 7 = 4
02 & 7 = 2
```

compare to needed offset:

```
5
7
4
7
1
```

Maybe close, but no match.

Another idea:

could it be

```scss
```

```
(b << 3) + (c >> 5)
```

since `c` is 8-bit, its upper bits:

```r
c >> 5
8B >> 5 = 0x4
C0 >> 5 = 0x6
6E >> 5 = 0x3
DC >> 5 = 0x6
02 >> 5 = 0x0
```

then

```bash
(b << 3) + (c >> 5)
11*8 + 4 = 88 + 4 = 92 (vs 93)
8*8 + 6 = 64 + 6 = 70 (vs 71)
2*8 + 3 = 16 + 3 = 19 (vs 20)
1*8 + 6 = 8 + 6 = 14 (vs 15)
6*8 + 0 = 48 + 0 = 48 (vs 49)
```

That is VERY close! Off by 1 in every case.

So the rule:

```ini
out_shifted = (b << 3) + (c >> 5) + 1
```

and then shift left 12:

```ini
out = ((b << 3) + (c >> 5) + 1) << 12
```

Verification:

first row:

```bash
(11 << 3) + (139 >> 5) + 1
= 88 + 4 + 1 = 93
93 << 12 = 0x5D000 ✅
```

second row:

```bash
(8 << 3) + (192 >> 5) + 1
= 64 + 6 + 1 = 71
71 << 12 = 0x47000 ✅
```

third row:

```bash
(2 << 3) + (110 >> 5) + 1
= 16 + 3 + 1 = 20
20 << 12 = 0x14000 ✅
```

fourth row:

```bash
(1 << 3) + (220 >> 5) + 1
= 8 + 6 + 1 = 15
15 << 12 = 0xF000 ✅
```

fifth row:

```bash
(6 << 3) + (2 >> 5) + 1
= 48 + 0 + 1 = 49
49 << 12 = 0x31000 ✅
```

Found it!

✅ Answer

The formula is:

$$\text{out} = ((b \ll 3) + (c \gg 5) + 1) \ll 12$$

where `b` and `c` are 8-bit hex inputs.

If you want, I can help you phrase that for Wolfram Alpha too — just ask!

give me the rust code of the formula

Sure — here's the equivalent Rust code for that formula:

```rust
fn calculate_out(b: u8, c: u8) -> u32 {
    let out_shifted = ((b as u32) << 3) + ((c as u32) >> 5) + 1;
    out_shifted << 12
}

fn main() {
    let test_cases = [
        (0x0B, 0x8B), // should be 0x05D000
        (0x08, 0xC0), // should be 0x047000
        (0x02, 0x6E), // should be 0x014000
        (0x01, 0xDC), // should be 0x00F000
        (0x06, 0x02), // should be 0x031000
    ];

    for (b, c) in test_cases {
        let out = calculate_out(b, c);
        println!("b=0x{:02X}, c=0x{:02X} -> out=0x{:06X}", b, c, out);
    }
}
```

If you run this, it will print the matching results in hex for easy verification.

If you'd like, I can help break this down line by line — just let me know!