# Section 3 / Bit Fields / With Bit Fields

Given how long the previous chapter, describing life without bit fields, was this chapter will be a let down.

Recall:

```
struct BF {
    unsigned char a : 1;
    unsigned char b : 2;
    unsigned char c : 5;
};
```

With bit fields, assigning values to `a`, `b` and `c` are just:

```
bf.a = 1;
bf.b = 2;
bf.c = 3;
```

What about ensuring the values to be assigned to the bit fields are within the right range? For example:

```
bf.c = 345;
```

Clearly, 345 cannot fit in 5 bits. Depending upon the compiler you should get a warning or an error. For example:

```
test.c:61:12: warning: unsigned conversion from ‘int’ to ‘volatile unsigned char:5’ changes
   61 |     bf.c = 345;
      |
                  ^~~
```

As for the assembly language that bit field will produce, it depends upon optimization level. Unoptimized, the code produced will be much longer and cumbersome than the “sophisticated” assembly language in the previous chapter.

Turning on full optimization, you’re get pretty much the same assembly language generated as the sophisticated version in the previous chapter.

One remaining difference is that the optimized bit field instructions will not require the initial `ldr` nor the final `str` to load and store the byte containing our fields.

Note that the C version of the bit field setters could be declared to be `inline` so that would be closer in performance to equivalent code written to use C / C++ bit fields.