To compute an n-bit binary CRC, line the bits representing the input in a row, and position the (n + 1)-bit pattern representing the CRC's divisor (called a "polynomial") underneath the left end of the row.

In this example, we shall encode 14 bits of message with a 3-bit CRC, with a polynomial x3 + x + 1. The polynomial is written in binary as the coefficients; a 3rd-degree polynomial has 4 coefficients (1x3 + 0x2 + 1x + 1). In this case, the coefficients are 1, 0, 1 and 1. The result of the calculation is 3 bits long, which is why it is called a 3-bit CRC. However, you need 4 bits to explicitly state the polynomial.

Start with the message to be encoded:

```
11010011101100
```

This is first padded with zeros corresponding to the bit length n of the CRC. This is done so that the resulting code word is in systematic form. Here is the first calculation for computing a 3-bit CRC:

11010011101100 000 <--- input right padded by 3 bits
1011 <--- divisor (4 bits) = $x^3 + x + 1$

01100011101100 000 <--- result

The algorithm acts on the bits directly above the divisor in each step. The result for that iteration is the bitwise XOR of the polynomial divisor with the bits above it. The bits not above the divisor are simply copied directly below for that step. The divisor is then shifted right to align with the highest remaining 1 bit in the input, and the process is repeated until the divisor reaches the right-hand end of the input row. Here is the entire calculation:

Input A	Input B	XOR Output
0	0	0
0	1	1
1	0	1
1	1	0

```
11010011101100 000 <--- input right padded by 3 bits
1011               <--- divisor
01100011101100 000 <--- result (note the first four bits are the XOR
with the divisor beneath, the rest of the bits are unchanged)
 1011              <--- divisor ...
00111011101100 000
  1011
00010111101100 000
   1011
00000001101100 000 <--- note that the divisor moves over to align with
the next 1 in the dividend (since quotient for that step was zero)
       1011             (in other words, it doesn't necessarily move
one bit per iteration)
00000000110100 000
        1011
00000000011000 000
         1011
00000000001110 000
          1011
00000000000101 000
           101 1
-----------------
00000000000000 100 <--- remainder (3 bits).  Division algorithm stops
here as dividend is equal to zero.
```

Since the leftmost divisor bit zeroed every input bit it touched, when this process ends the only bits in the input row that can be nonzero are the n bits at the right-hand end of the row. These *n* bits are the remainder of the division step, and will also be the value of the CRC function (unless the chosen CRC specification calls for some postprocessing).

The validity of a received message can easily be verified by performing the above calculation again, this time with the check value added instead of zeroes. The remainder should equal zero if there are no detectable errors.

```
11010011101100 100 <--- input with check value
1011               <--- divisor
01100011101100 100 <--- result
 1011              <--- divisor ...
00111011101100 100

......

00000000001110 100
          1011
00000000000101 100
           101 1
-----------------
00000000000000 000 <--- remainder
```