

Question 1

I wrote code to calculate the bytes of the AES S-Box. The intermediate values for the calculation of 10110010 are included in the description of the code.

```
polyAdd :: Word a => a -> a -> a
polyAdd = (.^.)

polyMulXn :: Word a => a -> Int -> a
polyMulXn = (.<<.)

polyMul :: Word a => a -> a -> a
polyMul = polyMul' polyMulXn

polyMul' :: Word a => (a -> Int -> a) -> a -> a -> a
polyMul' f x y = foldl (.^.) 0 $ map (f x.(ws-)) $ filter (testBit y) [1..ws]
  where ws = wordSize x
```

`polyAdd` and `polyMul` implement addition of polynomials with coefficients mod 2 represented as bits in a word. Multiplication is done by multiplying the LHS by each term of the RHS and summing the results.

```
polyQuotRem :: Word a => a -> a -> (a,a)
polyQuotRem _ 0 = error "division by zero"
polyQuotRem x 1 = (x,0)
polyQuotRem 0 _ = (0,0)
polyQuotRem x y
  | diff < 0 = (0,x)
  | otherwise = ((1.<<.diff) .|. q',r')
  where
    diff = degree x - degree y
    q = y .<<. diff
    r = x .^. q
    (q',r') = polyQuotRem r y
    degree x = pred $ (wordSize x-)
              $ fromJust $ find (testBit x) [1..wordSize x]
```

`polyQuotRem` calculates the quotient and remainder when one polynomial is divided by another. This function uses a process very similar to long division. The process stops when the degree of the remainder is less than the degree of the divisor. When this isn't the case the divisor is multiplied by X^n (where n is the difference in the degrees) and subtracted from the remainder (or the dividend on the first iteration). X^n is added to the quotient and the process continues.

```

polyExtEuclid :: Word a => a -> a -> (a,a,a)
polyExtEuclid a 0 = (1,0,a)
polyExtEuclid a b = (y,x `polyAdd` (y`polyMul`q),thegcd)
  where (x,y,thegcd) = polyExtEuclid b r
        (q,r) = polyQuotRem a b

```

polyExtEuclid implements the extended euclidian algorithm for polynomials. It is exactly the same as the standard extended euclidian algorithm except it uses the poly* functions. ¹

```

polyMultInvMod b n
  | thegcd == 1 = x
  | otherwise = error (polyShow x ++ " isn't irreducible")
  (x,_,thegcd) = polyExtEuclid n b

```

polyMultInvMod find the multiplicative inverse of a polynomial mod an irreducible polynomial. ²

```

gf8Add :: Word8 -> Word8 -> Word8
gf8Add = polyAdd

gf8MulX :: Word8 -> Word8
gf8MulX x
  | testBit x 1 = x' .^. 27 -- X^4 + X^3 + X + 1, the shift took care of X^8
  | otherwise = x'
  where x' = x .<<. 1

gf8Mul :: Word8 -> Word8 -> Word8
gf8Mul x = polyMul' (\n -> (iterate gf8MulX n!!)) x

gf8MultInv :: Word8 -> Word8
-- 283 == X^8 + X^3 + X + 1
gf8MultInv x = fromWord $ polyMultInvMod (fromWord x) (283::Word16)

```

gf8Add, gf8Mul, and gf8MultInv implement addition, multiplication, and inversion for the field $GF(2^8) \text{ mod } X^8 + X^4 + X^3 + X + 1$.

The inverse of $X^7 + X^5 + X^4 + X$ (10110010) is $X^4 + X^3 + X^2 + X + 1$ (00011111).

```

aesSBoxMatrix :: [[Bool]]
aesSBoxMatrix = map (take 8 . flip drop s) [7,6..0]
  where s = cycle $ map toEnum [0,0,0,1,1,1,1,1]

```

aesSBoxMatrix is the AES S-Box matrix. This matrix is constructed by rotating the bit pattern 10001111 to the left. When the S-Box matrix is multiplied by the vector [11111000] (the bits of the inverse of 10110010 in reverse order) the result is [00101010]

¹We could actually create a type class for polynomials and extEuclid in Brianweb.Math would work unmodified... that seems like overkill for this homework though.

²multInvMod would also work unmodified with a polynomial type class.

```
aesColumnVector :: [[Bool]]
aesColumnVector = map (:[]).toEnum [1,1,0,0,0,1,1,0]
```

aesColumnVector is the AES S-Box column vector. It is a matrix with a single column containing the bits 11000110. When the column vector is added to the vector [00101010] the result is [11101100]. Converted back to a polynomial this is $X^5 + X^4 + X^2 + X + 1$, or 00110111.

```
aesSBoxByte :: Word8 → Word8
aesSBoxByte x = foldl (.|.) 0 $ [bit a|(a,b:_)←zip [8,7..1] z,b]
  where
    inv = if x == 0 then 0 else gf8MultInv x
    y = map (:[]).testBit inv [8,7..1]
    z = aesSBoxMatrix `matrixMul` y `matrixAdd` aesColumnVector
```

aesSBoxByte calculates the values of byte x of the AES S-Box. First we find the inverse of the position in $GF(2^8)$ (or 0 if x is 0). Then we multiply the vector created from that by the S-Box matrix and add the column vector. Finally the resultant vector is turned back into a word.

map aesSBoxByte [0..255] will generate the entire AES S-Box.