multiple values for X and Y were tested and both implementations returned correct value each time. The Instruction Count for optimized version is 87 while the instruction count for non-optimized is 96.

Non optimized implementation (ie restoring division):

- initialization
  - initialize the quotient register to zero.
  - initialize the left-side of the divisor register with the divisor itself and right-side with zeros
  - initialize reminder register with the dividend
- we iterate a loop 9 times, and during each iteration, we subtract the divisor from remainder and check the result.
  - if result of the subtraction is positive, then we shift the quotient to left one time, then set Least-Significant-Bit (LSB) to 1.
  - otherwise, if the result is negative, then we undo the subtraction operation that we did, then shift to quotient one bit to left (but we keep LSB as 0).
- also while we are the loop, we right-shift divisor register.

Optimized implementation (ie nonrestoring division):

- initialization
  - initialize the left-side of the divisor register with the divisor itself and right-side with zeros
  - initialize the right-side of remainder register with dividend

- we iterate a loop 8 times and during each iteration
  - if value in remainder register is non-negative, then we shift it to left then subtract the divisor from it
  - otherwise, we shift to left, then add the divisor to it
  - after that if remainder register is still non-negative, then we bit-wise OR the remainder register with value 1 (0x1).
- after exiting the loop, we check whether remainder is negative. If it's, then add divisor to it
- then we bitwise AND the remainder register with value 0xFF. the result will be our quotient
- finally to get the remainder, right-shift the remainder register 8 times.