Lab2: Optimized GCD

# 1. Objective

Write an assembly program to implement the Binary GCD algorithm.

# 2. Background

Read the article below that discusses the Euclidean algorithm and Binary GCD (Stein’s algorithm).

Euclidean algorithm for computing the greatest common divisor: <https://cp-algorithms.com/algebra/euclid-algorithm.html>

Toy example: compute gcd(48, 18) with both methods.

Euclidean algorithm (mod-based)

* Start with a=48, b=18. Replace (a, b) with (b, a mod b) until b=0.
* 48 mod 18 = 12 ⇒ (a, b) = (18, 12).
* 18 mod 12 = 6 ⇒ (a, b) = (12, 6).
* 12 mod 6 = 0 ⇒ (a, b) = (6, 0).
* Stop; gcd = 6.

Binary GCD (Stein’s algorithm, shift/subtract)

* Start u=48, v=18.
* Both even: factor 2 repeatedly. Common power of two = 2 since 48=16×3 and 18=2×9 share one 2. Record shift=1; divide both by 2 once ⇒ u=24, v=9.
* Make both odd: u even, v odd ⇒ u >>= ctz(u) (right shift u by ctz(u) bit positions). 24→3 (divide by 8) ⇒ u=3, v=9.
* Both odd: replace the larger by the difference and make even again.
  + v>u ⇒ v = v − u = 9 − 3 = 6; make odd: 6→3 (divide by 2 twice) ⇒ v=3.
* Now u=3, v=3 ⇒ v − u = 0 ⇒ v=0.
* Stop; gcd odd part = u = 3. Restore common factor: 3 << shift = 3 × 2 = 6.

Both algorithms return 6, but Euclid uses division/modulo each step, while binary GCD uses only shifts, subtraction, and comparisons, which can be faster on hardware where division is expensive.

## How to implement ctz()

ctz() stands for “count trailing zeros”. It returns the number of consecutive 0-bits at the least-significant end of an integer’s binary representation. For example, ctz(48) = 4 because 48 = 0b0011 0000 ends with four zeros, and ctz(18) = 1 because 18 = 0b10010 ends with one zero. In the binary GCD (Stein’s) algorithm, ctz(x) gives the largest power of 2 dividing x, so dividing by 2^ctz(x) quickly removes all factors of two.)

ARMv7 does not have a native CTZ instruction, but you can implement it with bit-reverse + CLZ “count leading zeros.”.

ctz(x) = clz(rbit(x)); again handle x=0 separately.

For nonzero x, ctz(x) = clz(rbit(x)) because reversing the bits turns trailing zeros into leading zeros. Using x = 0b0010 1100 (44), rbit over 8 bits yields 0b0011 0100, which has 2 leading zeros, so ctz(x) = 2.

This trick only works for non-zero input x, since many implementations of ctz/clz consider input x = 0 as undefined behavior for performance reasons. So please define ctz(0)=32 explicitly before calling ctz(x) for x!=0.

# 3. Lab Steps

Start with the Assembly program below that implements the Euclidean algorithm, modify it to implement the Binary GCD algorithm.

Computing the Euclidean Algorithm in raw ARM Assembly

<https://www.youtube.com/watch?v=665rzOSSxWA>

<https://github.com/LaurieWired/Assembly-Algorithms/tree/main/GCD>

# 5 Report

Please use the project report template and submit the report in PDF format. Submit a separate source file for the Binary GCD algorithm.