Lab2: GCD Algorithm

1. Objective

Write assembly programs to implement the 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/Euclidean-algorithm.html

Video tutorials:

Recursive Euclidean: GCD - Euclidean Algorithm (Method 2):
https://www.youtube.com/watch?v=svBx8u5bMEg

Non-recursive Euclidean GCD - Euclidean Algorithm (Method 1):
https://www.youtube.com/watch?v=yHwneN6zJmU

We use a pair of data $a = 48$ and $b = 18$ for illustration. All three variants compute $\gcd(48,18) = 6$, but Euclidean (recursive or non-recursive) uses division/modulo each step, while binary GCD uses only shifts, subtraction, and comparisons, which can be faster on hardware where division is expensive.

Recursive Euclidean

```
// Recursive GCD
int gcd (int a, int b) {
  if (b == 0)
    return a;
  else
    return gcd (b, a % b);
}
```

- Start with $gcd(48,18)$. Since $18 \neq 0$, recurse to $gcd(18, 48 \bmod 18) = gcd(18,12)$.
- Next $gcd(18,12) \rightarrow gcd(12,18 \bmod 12) = gcd(12,6)$.
- Next $gcd(12,6) \rightarrow gcd(6,12 \bmod 6) = gcd(6,0)$.
- Base case returns 6, so $\gcd(48,18) = 6$.

Iterative (non-recursive) Euclidean

```
// Iterative GCD
int gcd (int a, int b) {
  while (b) {
    a %= b;
    swap(a, b);
  }
  return a;
}
```

- 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)

```
// Binary GCD (Stein's algorithm)
// Computes gcd(a, b) using only shifts, subtraction, and comparisons.
int gcd(int a, int b) {
  // If either is zero, GCD is the bitwise OR (the other operand).
  // This works because if a==0, a|b == b; if b==0, a|b == a.
  if (!a || !b)
    return a | b;  // handles (0, x) and (x, 0) in O(1)

  // shift = number of common powers of two dividing both a and b.
  // ctz(x) = count trailing zeros in binary; ctz(a|b) gives min(ctz(a), ctz(b)).
  unsigned shift = __builtin_ctz(a | b);  // factor 2^shift out and restore at the end

  // Make a odd by removing all factors of two.
  a >>= __builtin_ctz(a);  // divide a by 2^ctz(a)

  // Main loop: maintain a odd; reduce b until it becomes zero.
  do {
    // Remove all factors of two from b to make it odd as well.
    b >>= __builtin_ctz(b);  // divide b by 2^ctz(b)

    // Ensure a <= b to keep subtraction non-negative.
    if (a > b)
      swap(a, b);  // now a <= b

    // Replace (a, b) with (a, b - a); gcd is invariant under subtracting equals for odd a, b.
    b -= a;  // b becomes even (difference of two odds), next iteration will strip the 2s
  } while (b);  // stop when b hits 0; then a holds gcd without the common 2^shift factor

  // Restore the common power-of-two factor that was factored out initially.
  return a << shift;  // multiply gcd by 2^shift to get final result
}
```

- Start $(a, b) = (48{,}18)$. If either is zero, return the other; not the case. Compute common power-of-two factor: $a|b = 48|18 = 50$ has $\text{ctz}(50) = 1$, so extract one factor of 2 at the end; set shift $= 1$. (48 in binary is 110000, and 18 is 010010; bitwise OR 48|18 gives 110010, which is decimal 50 with one trailing zero.)
- Remove factors of two individually: $a = 48$ has $\text{ctz}(48) = 4 \to a \leftarrow 48 \gg 4 = 3$; $b = 18$ has $\text{ctz}(18) = 1 \to b \leftarrow 18 \gg 1 = 9$.
- Now both odd. Repeat:
  - Ensure $a \leq b$; currently $(a, b) = (3{,}9)$. Set $b \leftarrow b - a = 9 - 3 = 6$, which is even.

  - Normalize b by removing factors of two: $\text{ctz}(6) = 1 \rightarrow b \leftarrow 6 \gg 1 = 3$.
  - Now $(a, b) = (3,3)$. Since $a \leq b$, set $b \leftarrow b - a = 3 - 3 = 0$.
- Loop ends at $b = 0$. Result before restoring powers of two is $a = 3$. Restore the common factor: return $a \ll \text{shift} = 3 \ll 1 = 6$, hence $\text{gcd}(48,18) = 6$.

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)) (handle the case of 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 Recursive Euclidean 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
Modify it to implement (1) the Iterative (non-recursive) Euclidean algorithm; (2) the Binary GCD algorithm.

5. Report

Use the project report template and submit the report in PDF format. Submit two separate source files for parts (1) and (2), with the input pair (48, 18).