Breakdown of `gcd.s`:

- Program starts here sequentially:

```
gcd:
    beq $a0, $a1, exit  # if a = b, go to exit
```

The code below recursively subtracts b from a until a < b.

- `beq`: branch on equal
    - If `$a0` (a) == `$a1` (b), go to `exit`

```
    slt $v0, $a1, $a0   # is b > a?
```

- `slt`: set less than
- `$v0`: comparison (function) result, which is either 0 or 1

```
    bne $v0, $0, suba   # if yes, goto suba
```

- `bne`: branch not equal
- `$0` = 0 for convenience
    - Branch to `suba` if `$v0` != 0

```
    subu $a0, $a0, $a1  # subtract b from a
```

- `subu`: subtract unsigned
- `$a0` -= `$b0`

```
    b gcd           # and repeat
```

- `b` branches back to `gcd`
- A new value of `$a0` is passed to `gcd`

```
suba:
    subu $a1, $a1, $a0  # subtract a from b
    b gcd               # and repeat
```

Same thing as above, except the order of subtraction swapped

```
exit:
    move $v0, $a0       # return a
```

- `$a0` (a) now holds final gcd value

```
    jr $ra              # go back to caller
```

- `jr` means "jump register" to `$ra`
- `$ra` is the of where gcd is **initially** called. It will not recursively exit to `slt $v0, $a1, $a0`
- The control ends here, and the output is at `$v0`