

**Faculty of Engineering & Technology**

**Electrical & Computer Engineering Department**

**COMPUTER ORGANIZATION AND ASSEMBLY LANGUAGE - ENCS336**

**Final Project Report**

**GCD and LCM and using them in adding fractions**

**Prepared by:**

Obada Tahayna 1191319

**Instructor**: Dr. Abdallatif Abuissa

**Section:** 1  
**Date:** 02/01/2021

# **Table of Contents**

[Table of Contents II](#_Toc60878968)

[Table of Figures III](#_Toc60878969)

[Program Overview 4](#_Toc60878970)

[Declaring 4](#_Toc60878971)

[Macros 4](#_Toc60878972)

[**printStr** 4](#_Toc60878973)

[**printNum** 4](#_Toc60878974)

[**scan** 5](#_Toc60878975)

[**printThreeDigit** 5](#_Toc60878976)

[**printFiveDigit** 6](#_Toc60878977)

[Code Area 7](#_Toc60878978)

[**Part1: Calculate GCD and LCM of Two Numbers** 7](#_Toc60878979)

[**Part2: Find the Result of Adding Two Fraction Using GCD** 8](#_Toc60878980)

[Procedures 11](#_Toc60878981)

[**getGCD** 11](#_Toc60878982)

[**scanThreeDigit** 13](#_Toc60878983)

[Outputs Samples 14](#_Toc60878984)

[Part1: Calculate GCD and LCM of Two Numbers 14](#_Toc60878985)

[Part2: Find the Result of Adding Two Fraction Using GCD 15](#_Toc60878986)

[Full Output 16](#_Toc60878987)

# **Table of Figures**

[Figure 1: Declaring 4](#_Toc60879055)

[Figure 2: printStr Macro 4](https://d.docs.live.net/b52ec1356bbb978b/Desktop/FinalProjectReport.docx#_Toc60879056)

[Figure 3: printNum Macro 4](https://d.docs.live.net/b52ec1356bbb978b/Desktop/FinalProjectReport.docx#_Toc60879057)

[Figure 4: scan Macro 5](https://d.docs.live.net/b52ec1356bbb978b/Desktop/FinalProjectReport.docx#_Toc60879058)

[Figure 5: printThreeDigit Macro 5](#_Toc60879059)

[Figure 6: printFiveDigit Macro 6](#_Toc60879060)

[Figure 7: The Code of Reading 2 Numbers 7](https://d.docs.live.net/b52ec1356bbb978b/Desktop/FinalProjectReport.docx#_Toc60879061)

[Figure 8: Code Area - Calculate GCD and LCM 7](#_Toc60879062)

[Figure 9: Code Area - Print GCD and LCM of the Two Numbers 8](https://d.docs.live.net/b52ec1356bbb978b/Desktop/FinalProjectReport.docx#_Toc60879063)

[Figure 10: Code for Reading Fractions 8](https://d.docs.live.net/b52ec1356bbb978b/Desktop/FinalProjectReport.docx#_Toc60879064)

[Figure 11: Code for Making the Denominators the Same 9](https://d.docs.live.net/b52ec1356bbb978b/Desktop/FinalProjectReport.docx#_Toc60879065)

[Figure 12: Code for Adding Numerators and Check error 9](https://d.docs.live.net/b52ec1356bbb978b/Desktop/FinalProjectReport.docx#_Toc60879066)

[Figure 13: Code for Simplifying the Result and Print it 10](https://d.docs.live.net/b52ec1356bbb978b/Desktop/FinalProjectReport.docx#_Toc60879067)

[Figure 14: Example of finding GCD using Euclid's Algorithm 11](https://d.docs.live.net/b52ec1356bbb978b/Desktop/FinalProjectReport.docx#_Toc60879068)

[Figure 15: getGCD Procedure Code 12](https://d.docs.live.net/b52ec1356bbb978b/Desktop/FinalProjectReport.docx#_Toc60879069)

[Figure 16: scanThreeDigit Procedure Code 13](#_Toc60879070)

[Figure 17: Sample of Part1 Outputs (With Error Messages) 14](https://d.docs.live.net/b52ec1356bbb978b/Desktop/FinalProjectReport.docx#_Toc60879071)

[Figure 18: Sample of Part1 Outputs (Without Invalid Inputs) 14](#_Toc60879072)

[Figure 19: Sample of Part2 Outputs (With Invalid Input Errors) 15](#_Toc60879073)

[Figure 20: Sample of Part2 Outputs (With Big Number Error) 15](#_Toc60879074)

[Figure 21: Full Output of The Program 16](#_Toc60879075)

# **Program Overview**

## **Declaring**

First of all, I declared the capacity of the model (small), and the stack size (100), and then the variables declared.

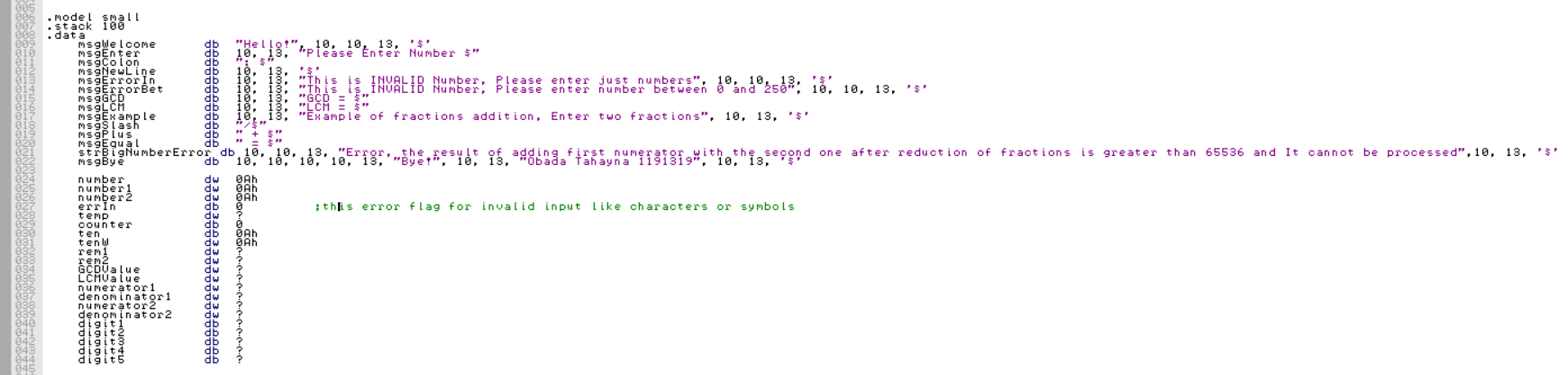
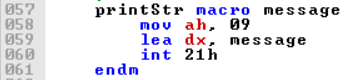


Figure : Declaring

## **Macros**

After the variables grabbed to ds and the value of ax made to be zero

then the macros started.



### **printStr**

Figure : printStr Macro

This macro is to print a string message.

### **printNum**

Figure : printNum Macro

Macro to print a number of one digit.

### **scan**

Figure : scan Macro

Scan macro to get a one-digit number from user using int 21h.

### **printThreeDigit**

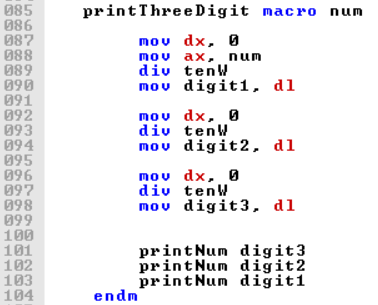


Figure : printThreeDigit Macro

Macro that prints any 3-digit number, this macro built on printNum macro, by split the number to its digits and print each digit using printNum macro.

### **printFiveDigit**

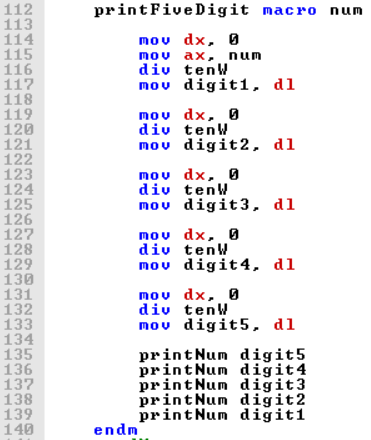


Figure : printFiveDigit Macro

Like printThreeDigit macro, printFiveDigit Macro uses printNum macro to print the 5 digits of the 5-digit number.

## **Code Area**

### **Part1: Calculate GCD and LCM of Two Numbers**

#### **Read Two Numbers**

Loop to take 2 numbers from 3 digits and save them in the memory at first, then the values of 2 numbers taken from memory to variables (number1 and number2) after exiting the loop.

In the procedure scanThreeDigits, if any digit isn’t a number, the value of the errIn flag variable becomes to be 1 ([see how](#_scanThreeDigit)), and here in the code when we get the two numbers, if any number has errIn equals 1 then that means there is an error with this number, so the program asks the user to enter it again.

If all digits are numbers (errIn is 0), then the program checks if the number entered is greater then 250, if so, an error message will show on the screen and the program will ask the user to enter the number again.

Figure : The Code of Reading 2 Numbers

#### **Calculate GCD and LCM**

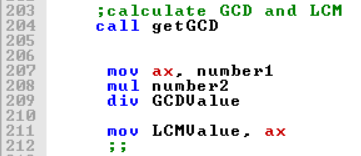


Figure : Code Area - Calculate GCD and LCM

The program calls [getGCD procedure](#_getGCD) that calculates the GCD of number1 and number2, then the result of LCM calculated as follow: LCM = (number1\*number2)/GCD.

#### **Print the GCD and LCM of the Two Numbers**

Figure : Code Area - Print GCD and LCM of the Two Numbers

The values of GCD and LCM printed after calculate them as the code mentioned.

### **Part2: Find the Result of Adding Two Fraction Using GCD**

#### **Reading Fractions**

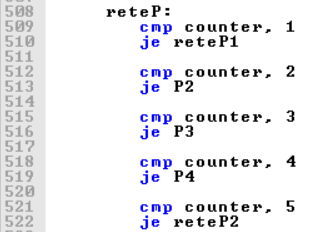


Figure : Code for Reading Fractions

The code that reads fractions looks very complex, the complexity of the code is because the way that the code handles with input errors (such as entering characters or symblos or entering numbers greater than 250).

In this part, the program will read 4 integers, first numerator, first denominator, second numerator and the second denominator. If the user entered invalid input in any of the four numbers, the porgram will show an error message and will ask the user to enter the number again after displaying the previous entered numebrs. ([see the outputs](#_Part2:_Find_the))

#### 

#### **Calculations**

##### **Making the Denominators the Same**

First, the program will calculate the GCD of the first denominator and the second one, Then it calculates the deviation of each denominators by the GCD calculated to find two values and store them on rem1 and rem2, After that the program multiplies numerator1 with rem2 and numerator2 with rem1 and demonator1 with rem2.

Figure : Code for Making the Denominators the Same

##### **Adding Numerators After Making the Denominators the Same**

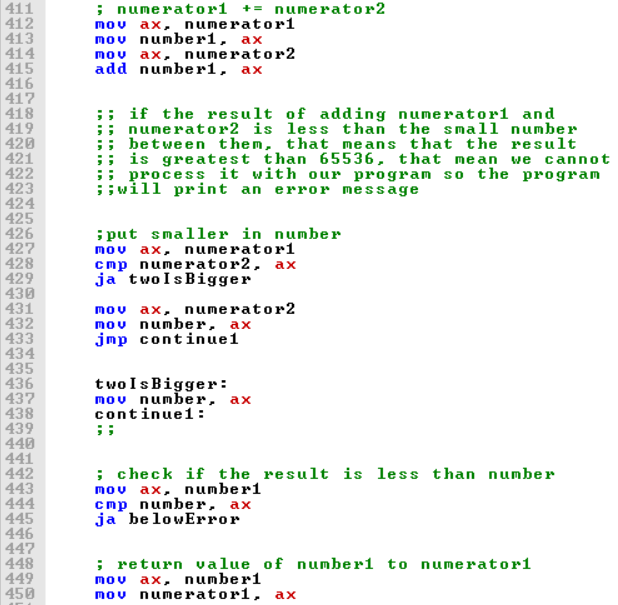


Figure : Code for Adding Numerators and Check error

After the denominators of the two fractions became same, the program will add the first numerator and the second one as the numerator of result fraction, if the result of adding the numerators is greater than 65536 (more the 16 bit), then the program cannot process that number because we used 16-bit registers, so if the result is greater than 16 bits, the program will display an error message.

If we add 2 numbers in 16-bit register, and the result of adding is more than 16 bits, then the result will be less than the lowest of the two numbers. ([Check the outputs](#_Part2:_Find_the))

##### **Simplifying the Result**

Figure : Code for Simplifying the Result and Print it

If there are no errors, the program will store the fraction result as numerator and denominator in numerator1 and denominator1. Then to simplify the result, the program finds the GCD of numerator and denominator, then we can find the simplest form by dividing each numerator and denominator by the DCG of them.

## **Procedures**

### **getGCD**

This procedure calculates the GCD of the two values stored on number1 and number2 variables and stores the result in GCDValue.

I used The Euclidean Algorithm to find the GCD of the two numbers, The Euclidean Algorithm is a is an efficient method for computing the greatest common divisor (GCD) of two integers (numbers), the largest number that divides them both without a remainder.[[1]](#footnote-1)

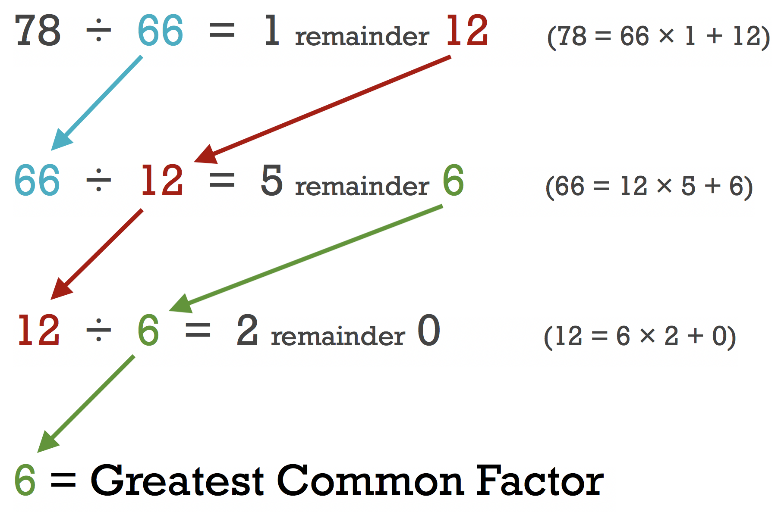
To use Euclid’s algorithm, divide the smaller number by the larger number. If there is a remainder, then continue by dividing the smaller number by the remainder.

A ÷ B = Q1 remainder R1

B ÷ R1 = Q2 remainder R2

R1 ÷ R2 = Q3 remainder R3

Continue this process until the remainder is 0 then stop. The divisor in the final step will be the greatest common factor.



For example, find the greatest common factor of 78 and 66 using Euclid’s algorithm.

78 ÷ 66 = 1 remainder 12

66 ÷ 12 = 5 remainder 6

12 ÷ 6 = 2 remainder 0

Thus, the greatest common factor is 6, since that was the divisor in the equation that yielded a remainder of 0. [[2]](#footnote-2)

Figure : Example of finding GCD using Euclid's Algorithm

For more information about Euclid’s algorithm, you can watch the video and visit the link below

Video: <https://www.youtube.com/watch?v=fwuj4yzoX1o>

Link: <https://www.khanacademy.org/computing/computer-science/cryptography/modarithmetic/a/the-euclidean-algorithm>

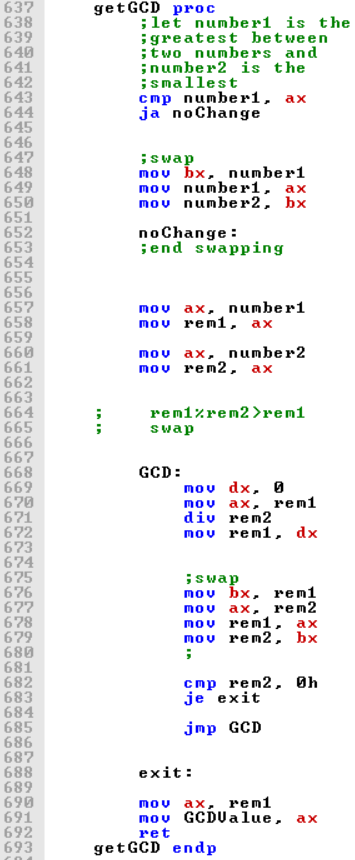


Figure : getGCD Procedure Code

First of all, the program checks which number is greater than the other one, and moves the greater number to number1 and the other to number2.

Then the value of rem1 set as number1, and the value of rem2 set as number2.

Our program will find the value of rem1 mod rem2, and then save it in rem1, then it will find rem2 mod the result of (rem1 mod rem2) that stored in rem1, so it will swap rem1 with rem2 (after store the last reminder in rem1).

This process will keep repeating until the result of mod is zero, when it is, the value of GCD will be the last value of rem1.

### **scanThreeDigit**

This procedure reads 3-digit numbers from user and save it in number variable.

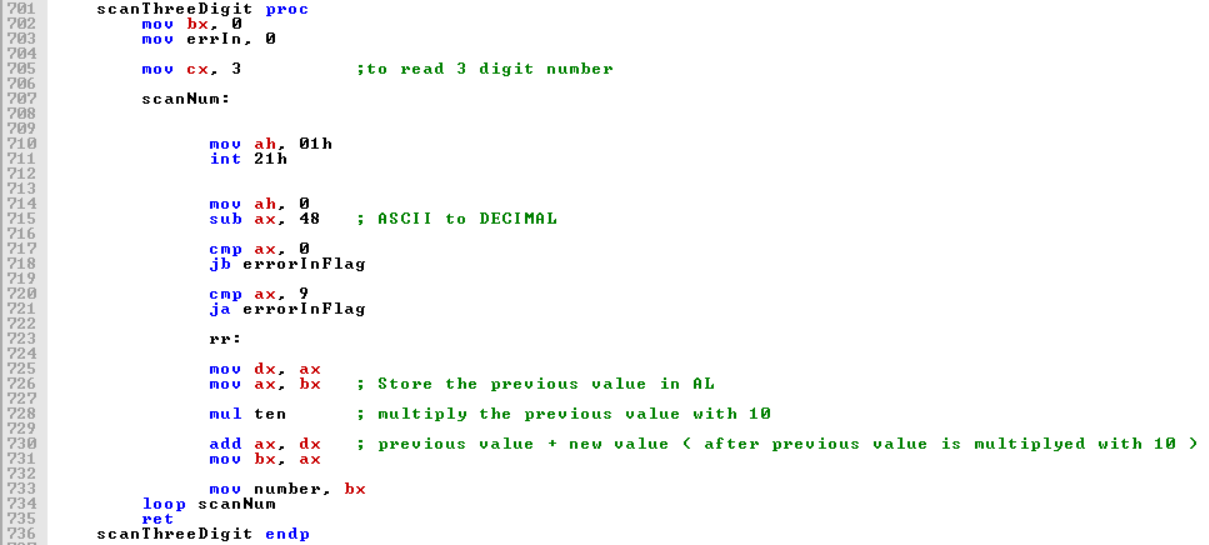


Figure : scanThreeDigit Procedure Code

First the value stored in cx set as the number of loops (equals the number of digits), in our case we need to read 3 digits so the value of cx set to 3.

The loop reads one digit each time. After read the digit, it will convert from ASCII to decimal, then the program checks if this digit isn’t number (its value is more than 9 or less than 0), if so, the errIn flag variable will set as 1.

Then to make a number from multi digits, the program saves the last value, then multiply it with 10 and add the new entry to it, this process repeated until cx become zero (all loops finished).

# **Outputs Samples**

## **Part1: Calculate GCD and LCM of Two Numbers**

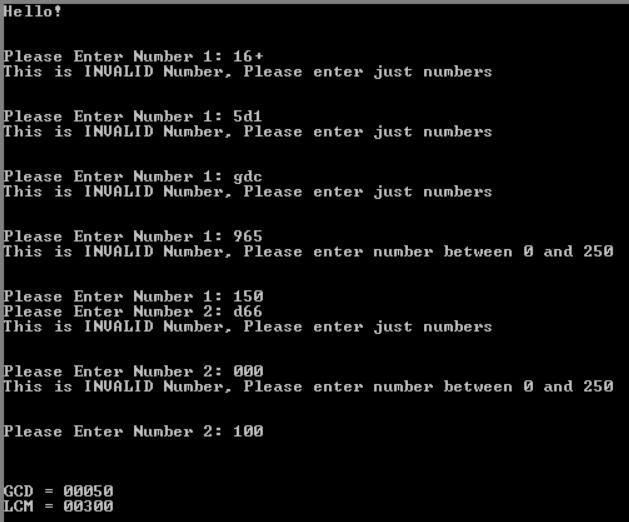


Figure : Sample of Part1 Outputs (With Error Messages)

As you see, if the user enters a special characters or letters, a message displays that it’s invalid number will appear, and asks the user to enter just numbers.

Also, if the user enters a number above 250, the program will display a message that tell the user that it’s invalid number because it’s above 250, and will ask user to enter number again, this will repeate until the user enters a valid number.

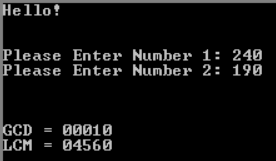


Figure : Sample of Part1 Outputs (Without Invalid Inputs)

## **Part2: Find the Result of Adding Two Fraction Using GCD**

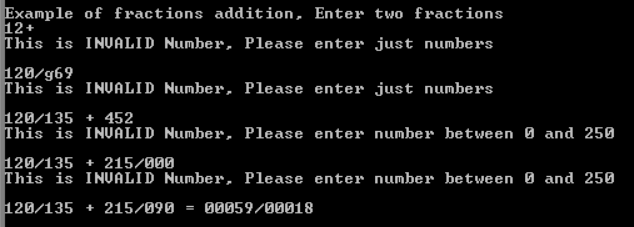


Figure : Sample of Part2 Outputs (With Invalid Input Errors)

The program asks user to enter fractions, user will enter numerator then denominator for the first then the second number respectively, if the user enter an invalid number, the program will display the previous numbers entered then it will wait user to enter the number again (Try the code for a better understanding).

If the user enters numbers like 211/223 + 199/227, the two denominators are prime numbers, that means when we try to make the denominators same, it will be 47897/50621 + 50621/50621, and when we try to add them, the result will be 98518/50621, but the problem is the numerator cannot store in 16-bit register because it’s too big, and that will cause an incorrect result, so the program will display an error message as follow.

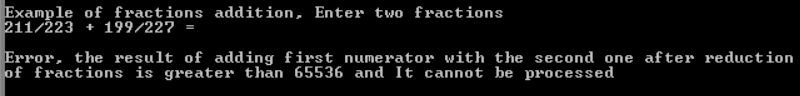


Figure : Sample of Part2 Outputs (With Big Number Error)

## **Full Output**

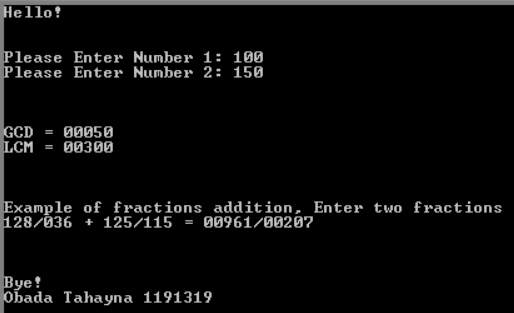


Figure : Full Output of The Program

1. [wikipedia.org](https://en.wikipedia.org/wiki/Euclidean_algorithm#:~:text=In%20mathematics%2C%20the%20Euclidean%20algorithm,them%20both%20without%20a%20remainder.) [↑](#footnote-ref-1)
2. [inchcalculator.com](https://www.inchcalculator.com/euclidean-algorithm-calculator/) [↑](#footnote-ref-2)