جامعة بيرزيت

BIRZEIT UNIVERSITY

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

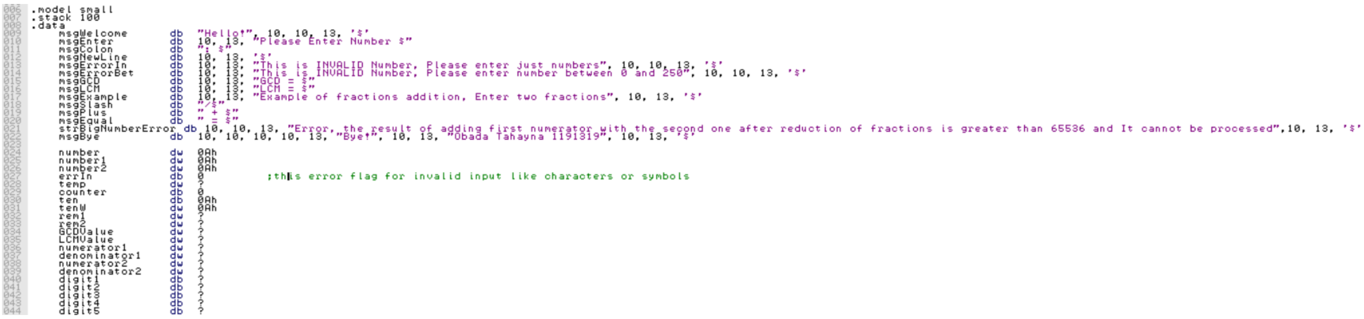
Table of Contents

Table of Figures

Program Overview

Declaring

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

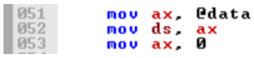


```
.model small
.stack 100
.data
    msgWelcome       db  "Hello!", 10, 10, 13, '$'
    msgEnter         db  10, 13, "Please Enter Number $"
    msgColon         db  ": $"
    msgNewLine       db  10, 13, '$'
    msgErrorIn       db  10, 13, "This is INVALID Number, Please enter just numbers", 10, 10, 13, '$'
    msgErrorBet      db  10, 13, "This is INVALID Number, Please enter number between 0 and 250", 10, 10, 13, '$'
    msgGCD           db  10, 13, "GCD = $"
    msgLCM           db  10, 13, "LCM = $"
    msgExample       db  10, 13, "Example of fractions addition, Enter two fractions", 10, 13, '$'
    msgSlash         db  "/$"
    msgPlus          db  " + $"
    msgEqual         db  " = $"
    strBigNumberError db 10, 10, 13, "Error, the result of adding first numerator with the second one after reduction of fractions is greater than 65536 and It cannot be processed",10, 13, '$'
    msgBye           db  10, 10, 10, 10, 13, "Bye!", 10, 13, "Obada Tahayna 1191319", 10, 13, '$'

    number           dw  0Ah
    number1          dw  0Ah
    number2          dw  0Ah
    errIn            db  0          ;this error flag for invalid input like characters or symbols
    temp             dw  ?
    counter          db  0
    ten              db  0Ah
    tenW             dw  0Ah
    rem1             dw  ?
    rem2             dw  ?
    GCDValue         dw  ?
    LCMValue         dw  ?
    numerator1       dw  ?
    denominator1     dw  ?
    numerator2       dw  ?
    denominator2     dw  ?
    digit1           db  ?
    digit2           db  ?
    digit3           db  ?
    digit4           db  ?
    digit5           db  ?
```

Figure 1: Declaring

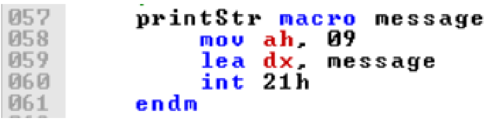
Macros

After the variables grabbed to ds and the value of ax made to be zero then the macros started.



```
mov ax, @data
mov ds, ax
mov ax, 0
```

printStr

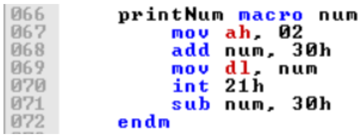


```
printStr macro message
    mov ah, 09
    lea dx, message
    int 21h
endm
```

Figure 2: printStr Macro

This macro is to print a string message.

printNum

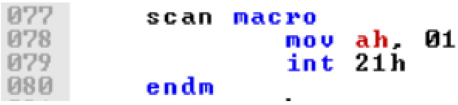


```
printNum macro num
    mov ah, 02
    add num, 30h
    mov dl, num
    int 21h
    sub num, 30h
endm
```

Figure 3: printNum Macro

Macro to print a number of one digit.

scan

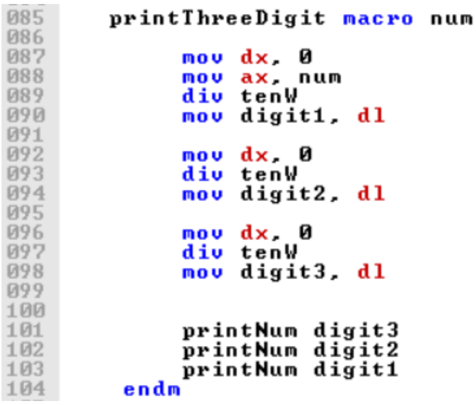


```
scan macro
        mov ah, 01
        int 21h
endm
```

Figure 4: scan Macro

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

printThreeDigit

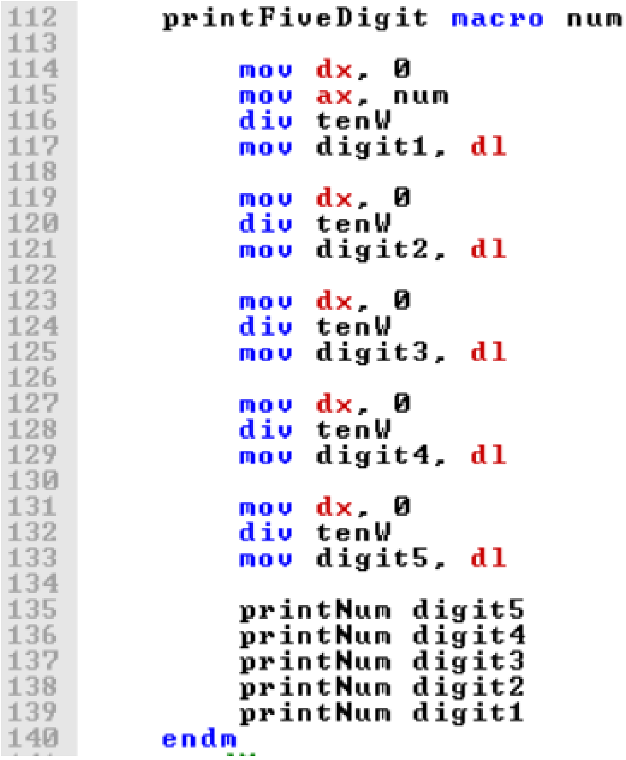


```
printThreeDigit macro num

    mov dx, 0
    mov ax, num
    div tenW
    mov digit1, dl

    mov dx, 0
    div tenW
    mov digit2, dl

    mov dx, 0
    div tenW
    mov digit3, dl


    printNum digit3
    printNum digit2
    printNum digit1
 endm
```

Figure 5: 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



```
printFiveDigit macro num

    mov dx, 0
    mov ax, num
    div tenW
    mov digit1, dl

    mov dx, 0
    div tenW
    mov digit2, dl

    mov dx, 0
    div tenW
    mov digit3, dl

    mov dx, 0
    div tenW
    mov digit4, dl

    mov dx, 0
    div tenW
    mov digit5, dl

    printNum digit5
    printNum digit4
    printNum digit3
    printNum digit2
    printNum digit1
endm
```

Figure 6: printFiveDigit Macro

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

Code Area

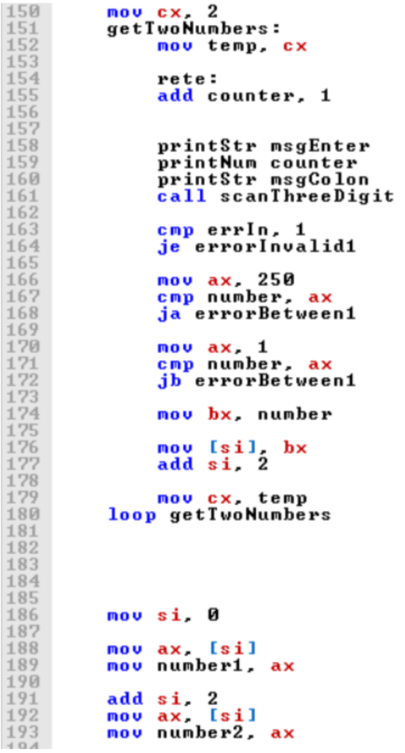
Part1: Calculate GCD and LCM of Two Numbers

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), 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.

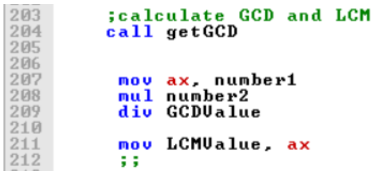
If all digits are numbers (errIn is 0), then the program checks if the number entered is greater then 250, if so, an error message will show on the screen and the program will ask the user to enter the number again.



```
    mov cx, 2
    getTwoNumbers:
        mov temp, cx

        rete:
        add counter, 1


        printStr msgEnter
        printNum counter
        printStr msgColon
        call scanThreeDigit

        cmp errIn, 1
        je errorInvalid1

        mov ax, 250
        cmp number, ax
        ja errorBetween1

        mov ax, 1
        cmp number, ax
        jb errorBetween1

        mov bx, number

        mov [si], bx
        add si, 2

        mov cx, temp
    loop getTwoNumbers





    mov si, 0

    mov ax, [si]
    mov number1, ax

    add si, 2
    mov ax, [si]
    mov number2, ax
```

Figure 7: The Code of Reading 2 Numbers

Calculate GCD and LCM



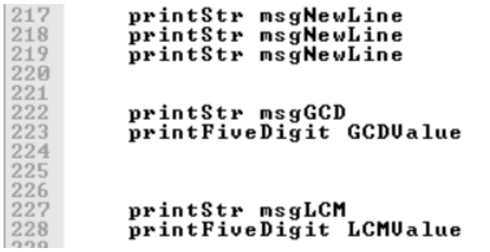
```
    ;calculate GCD and LCM
    call getGCD


     mov ax, number1
     mul number2
     div GCDValue

     mov LCMValue, ax
     ;;
```

Figure 8: Code Area - Calculate GCD and LCM

The program calls getGCD procedure 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

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

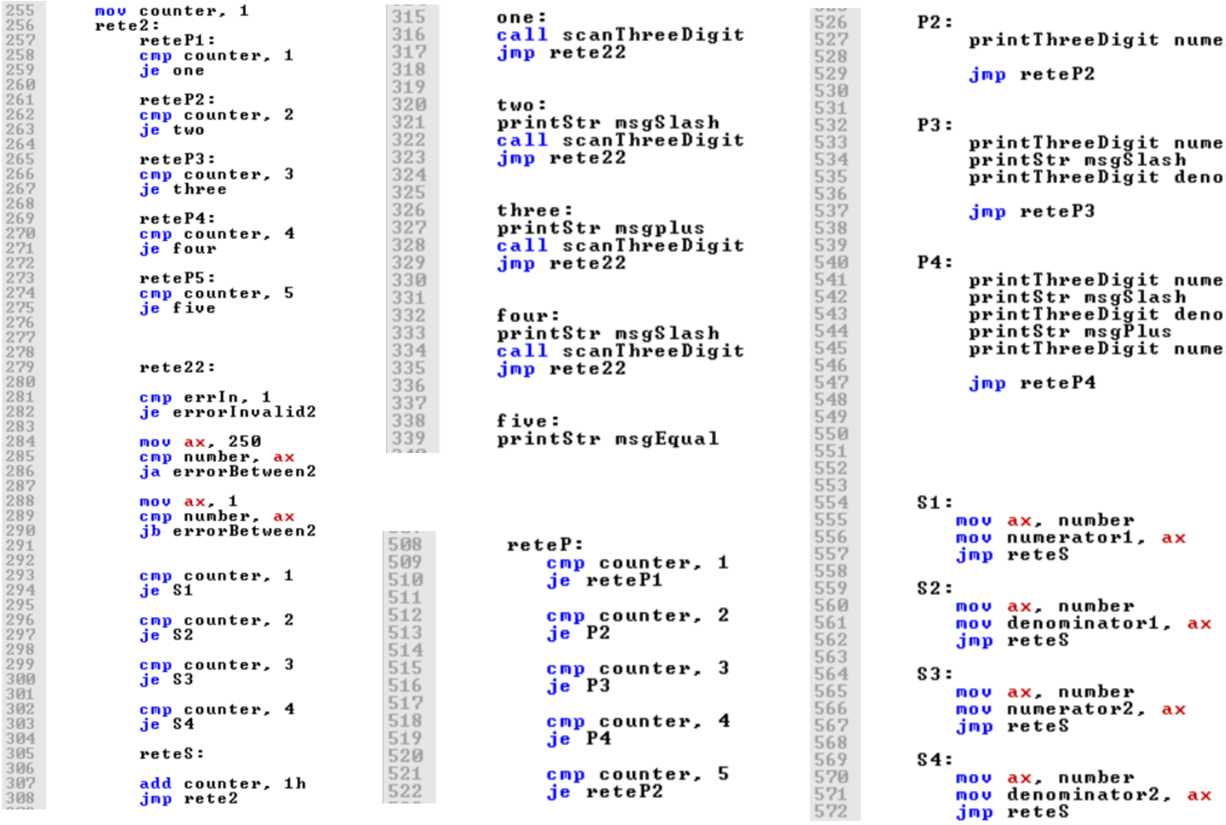


```
        printStr msgNewLine
        printStr msgNewLine
        printStr msgNewLine

        printStr msgGCD
        printFiveDigit GCDValue


        printStr msgLCM
        printFiveDigit LCMValue
```

Figure 9: Code Area - Print GCD and LCM of the Two Numbers

Part2: Find the Result of Adding Two Fraction Using GCD

Reading Fractions



```
mov counter, 1
rete2:
    reteP1:
    cmp counter, 1
    je one

    reteP2:
    cmp counter, 2
    je two

    reteP3:
    cmp counter, 3
    je three

    reteP4:
    cmp counter, 4
    je four

    reteP5:
    cmp counter, 5
    je five


    rete22:

    cmp errIn, 1
    je errorInvalid2

    mov ax, 250
    cmp number, ax
    ja errorBetween2

    mov ax, 1
    cmp number, ax
    jb errorBetween2

    cmp counter, 1
    je S1

    cmp counter, 2
    je S2

    cmp counter, 3
    je S3

    cmp counter, 4
    je S4

    reteS:

    add counter, 1h
    jmp rete2
```

```
one:
call scanThreeDigit
jmp rete22


two:
printStr msgSlash
call scanThreeDigit
jmp rete22


three:
printStr msgplus
call scanThreeDigit
jmp rete22


four:
printStr msgSlash
call scanThreeDigit
jmp rete22


five:
printStr msgEqual
```

```
 reteP:
    cmp counter, 1
    je reteP1

    cmp counter, 2
    je P2

    cmp counter, 3
    je P3

    cmp counter, 4
    je P4

    cmp counter, 5
    je reteP2
```

```
P2:
    printThreeDigit numerator1

    jmp reteP2


P3:
    printThreeDigit numerator1
    printStr msgSlash
    printThreeDigit denominator1

    jmp reteP3


P4:
    printThreeDigit numerator1
    printStr msgSlash
    printThreeDigit denominator1
    printStr msgPlus
    printThreeDigit numerator2

    jmp reteP4




S1:
   mov ax, number
   mov numerator1, ax
   jmp reteS

S2:
   mov ax, number
   mov denominator1, ax
   jmp reteS

S3:
   mov ax, number
   mov numerator2, ax
   jmp reteS

S4:
   mov ax, number
   mov denominator2, ax
   jmp reteS
```

Figure 10: Code for Reading Fractions

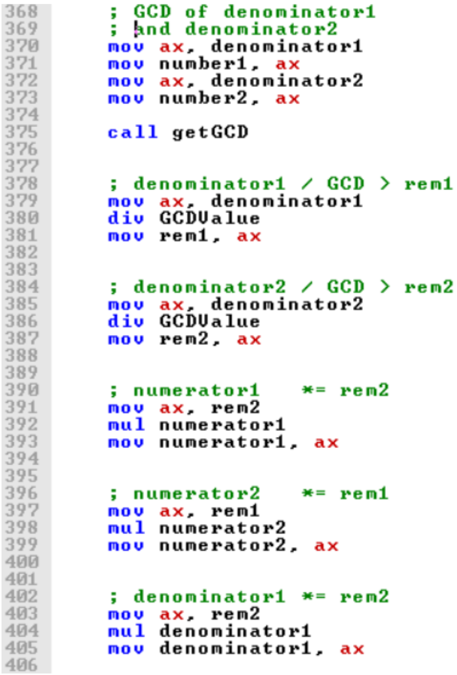
The code that reads fractions looks very complex, the complexity of the code is because the way that the code handles with input errors (such as entering characters or symblos or entering numbers greater than 250).

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)

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.



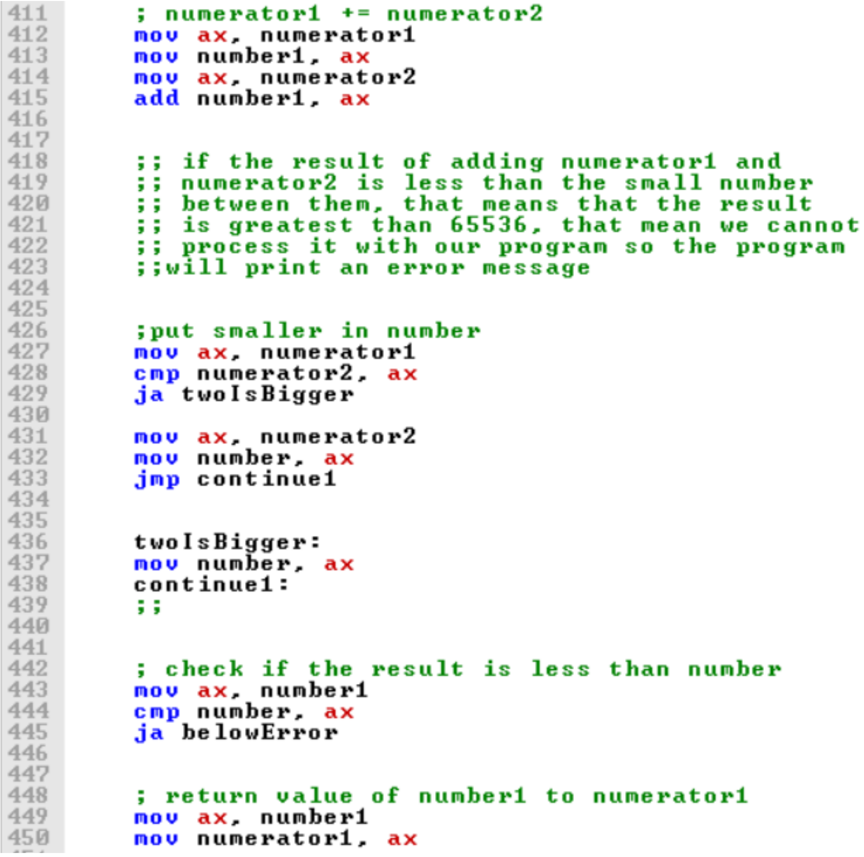
```
; GCD of denominator1
; and denominator2
mov ax, denominator1
mov number1, ax
mov ax, denominator2
mov number2, ax

call getGCD


; denominator1 / GCD > rem1
mov ax, denominator1
div GCDValue
mov rem1, ax


; denominator2 / GCD > rem2
mov ax, denominator2
div GCDValue
mov rem2, ax


; numerator1    *= rem2
mov ax, rem2
mul numerator1
mov numerator1, ax


; numerator2    *= rem1
mov ax, rem1
mul numerator2
mov numerator2, ax


; denominator1 *= rem2
mov ax, rem2
mul denominator1
mov denominator1, ax
```

Figure 11: Code for Making the Denominators the Same

Adding Numerators After Making the Denominators the Same

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)

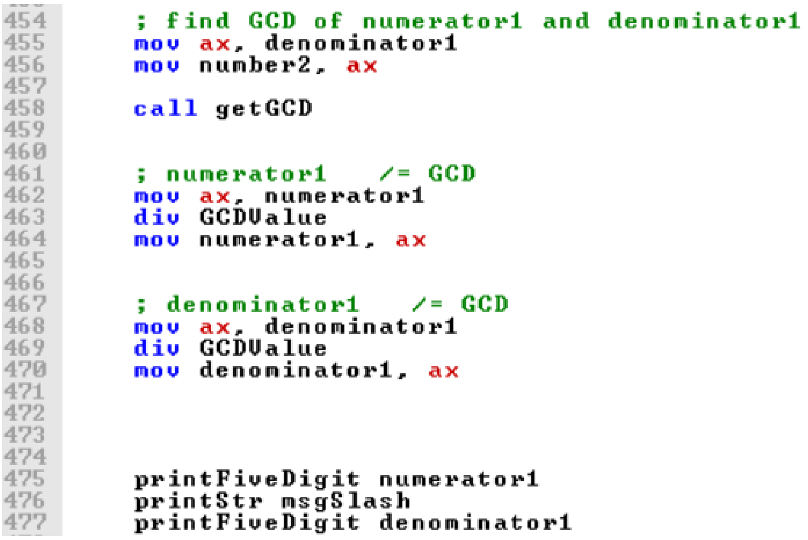


```
; numerator1 += numerator2
mov ax, numerator1
mov number1, ax
mov ax, numerator2
add number1, ax


;; if the result of adding numerator1 and
;; numerator2 is less than the small number
;; between them, that means that the result
;; is greatest than 65536, that mean we cannot
;; process it with our program so the program
;;will print an error message


;put smaller in number
mov ax, numerator1
cmp numerator2, ax
ja twoIsBigger

mov ax, numerator2
mov number, ax
jmp continue1


twoIsBigger:
mov number, ax
continue1:
;;


; check if the result is less than number
mov ax, number1
cmp number, ax
ja belowError


; return value of number1 to numerator1
mov ax, number1
mov numerator1, ax
```

Figure 12: Code for Adding Numerators and Check error

Simplifying the Result

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.



```
; find GCD of numerator1 and denominator1
mov ax, denominator1
mov number2, ax

call getGCD


; numerator1   /= GCD
mov ax, numerator1
div GCDValue
mov numerator1, ax


; denominator1   /= GCD
mov ax, denominator1
div GCDValue
mov denominator1, ax




printFiveDigit numerator1
printStr msgSlash
printFiveDigit denominator1
```

Figure 13: Code for Simplifying the Result and Print it

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]

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]

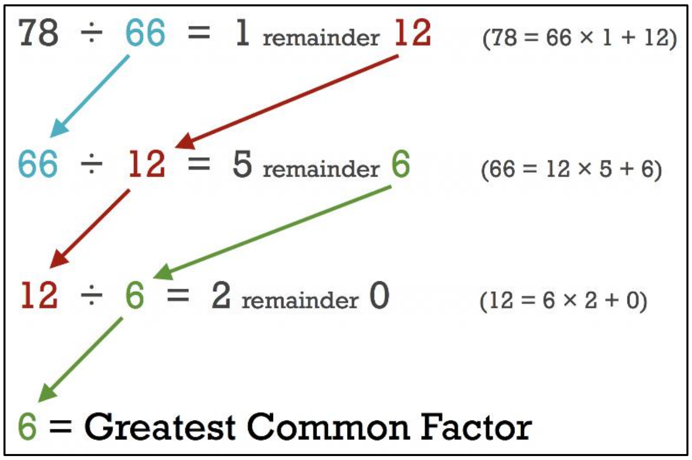


Figure 14: Example of finding GCD using Euclid's Algorithm

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

[1] wikipedia.org

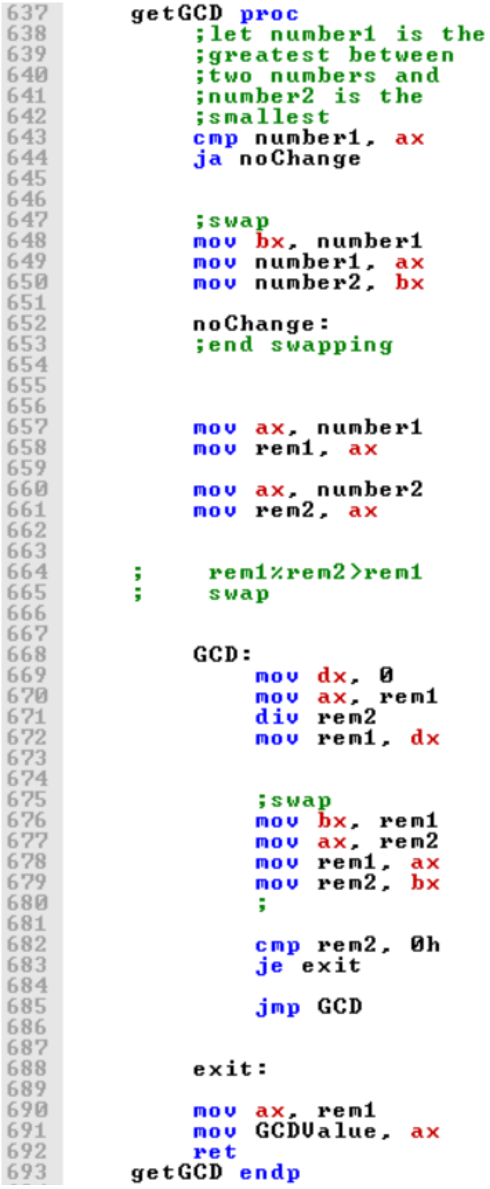
[2] inchcalculator.com

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.

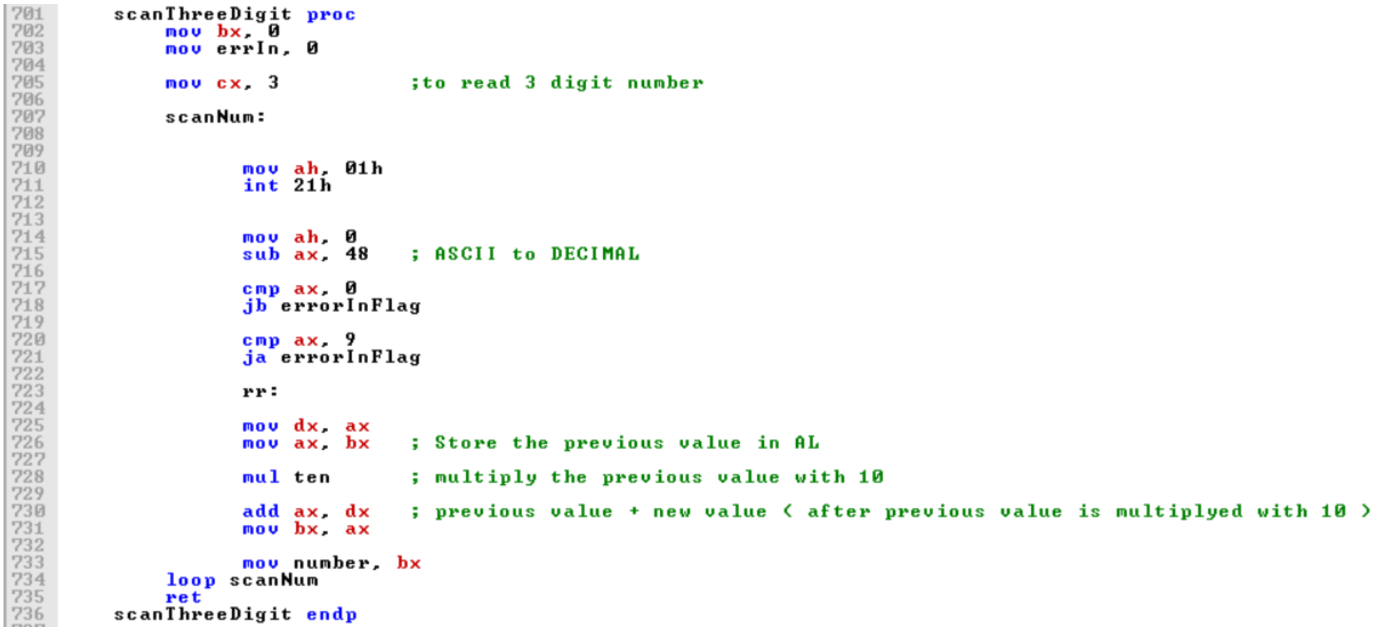


```
getGCD proc
    ;let number1 is the
    ;greatest between
    ;two numbers and
    ;number2 is the
    ;smallest
    cmp number1, ax
    ja noChange


    ;swap
    mov bx, number1
    mov number1, ax
    mov number2, bx

    noChange:
    ;end swapping



    mov ax, number1
    mov rem1, ax

    mov ax, number2
    mov rem2, ax


;    rem1%rem2>rem1
;    swap


    GCD:
        mov dx, 0
        mov ax, rem1
        div rem2
        mov rem1, dx


        ;swap
        mov bx, rem1
        mov ax, rem2
        mov rem1, ax
        mov rem2, bx
        ;

        cmp rem2, 0h
        je exit

        jmp GCD


    exit:

    mov ax, rem1
    mov GCDValue, ax
    ret
getGCD endp
```

Figure 15: getGCD Procedure Code

scanThreeDigit

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



```
scanThreeDigit proc
    mov bx, 0
    mov errIn, 0

    mov cx, 3              ;to read 3 digit number

    scanNum:


          mov ah, 01h
          int 21h


          mov ah, 0
          sub ax, 48   ; ASCII to DECIMAL

          cmp ax, 0
          jb errorInFlag

          cmp ax, 9
          ja errorInFlag

          rr:

          mov dx, ax
          mov ax, bx   ; Store the previous value in AL

          mul ten      ; multiply the previous value with 10

          add ax, dx   ; previous value + new value ( after previous value is multiplyed with 10 )
          mov bx, ax

          mov number, bx
    loop scanNum
    ret
scanThreeDigit endp
```

Figure 16: 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.

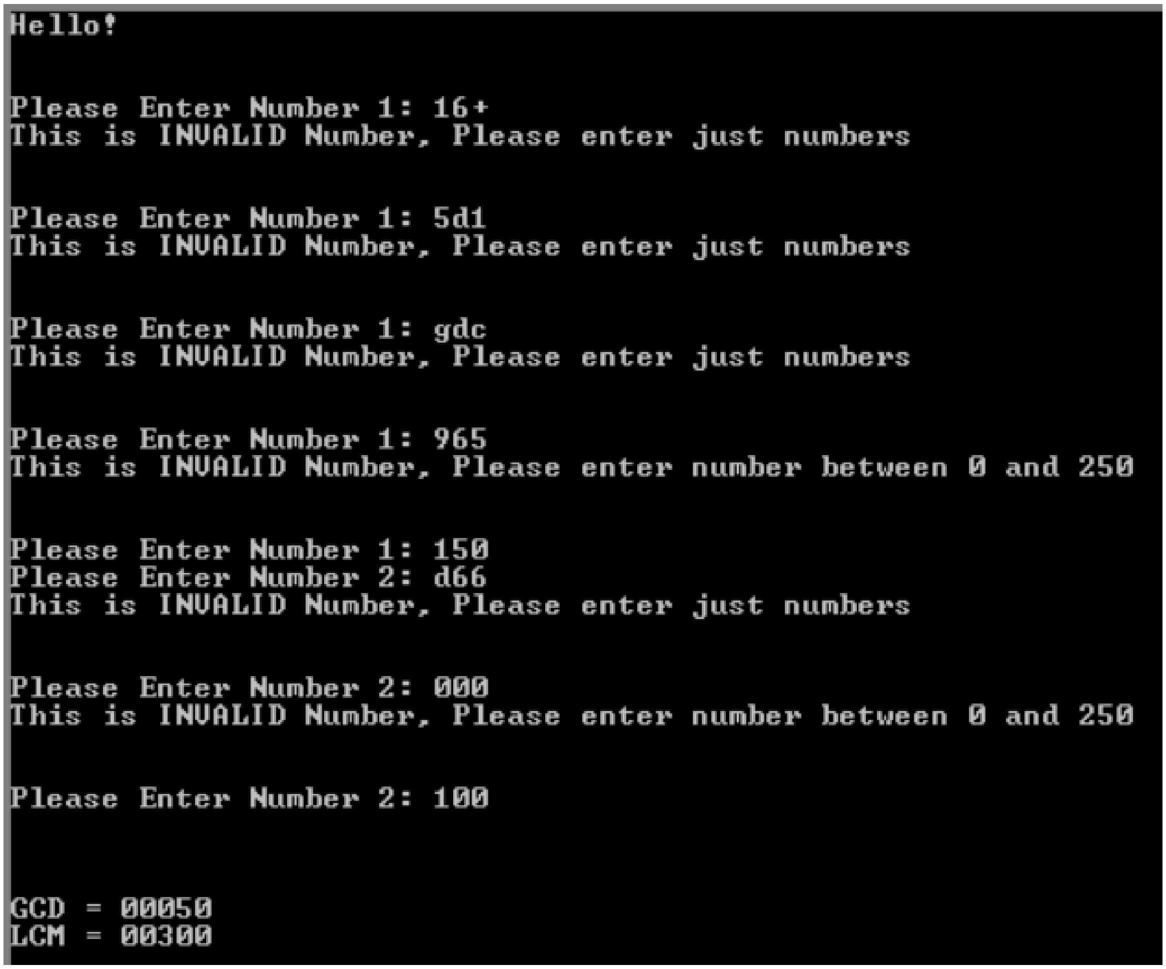
Then to make a number from multi digits, the program saves the last value, then multiply it with 10 and add the new entry to it, this process repeated until cx become zero (all loops finished).

Outputs Samples

Part1: Calculate GCD and LCM of Two Numbers

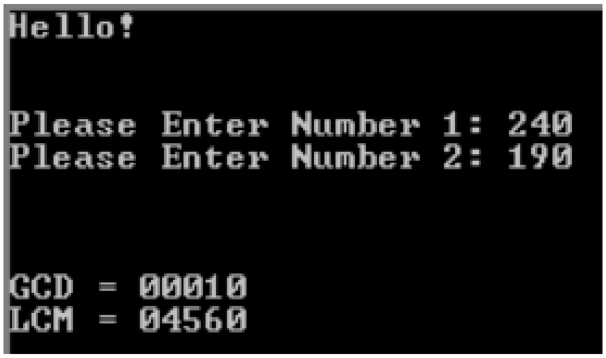
As you see, if the user enters a special characters or letters, a message displays that it's invalid number will appear, and asks the user to enter just numbers.

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.



```
Hello!


Please Enter Number 1: 16+
This is INVALID Number, Please enter just numbers


Please Enter Number 1: 5d1
This is INVALID Number, Please enter just numbers


Please Enter Number 1: gdc
This is INVALID Number, Please enter just numbers


Please Enter Number 1: 965
This is INVALID Number, Please enter number between 0 and 250


Please Enter Number 1: 150
Please Enter Number 2: d66
This is INVALID Number, Please enter just numbers


Please Enter Number 2: 000
This is INVALID Number, Please enter number between 0 and 250


Please Enter Number 2: 100



GCD = 00050
LCM = 00300
```

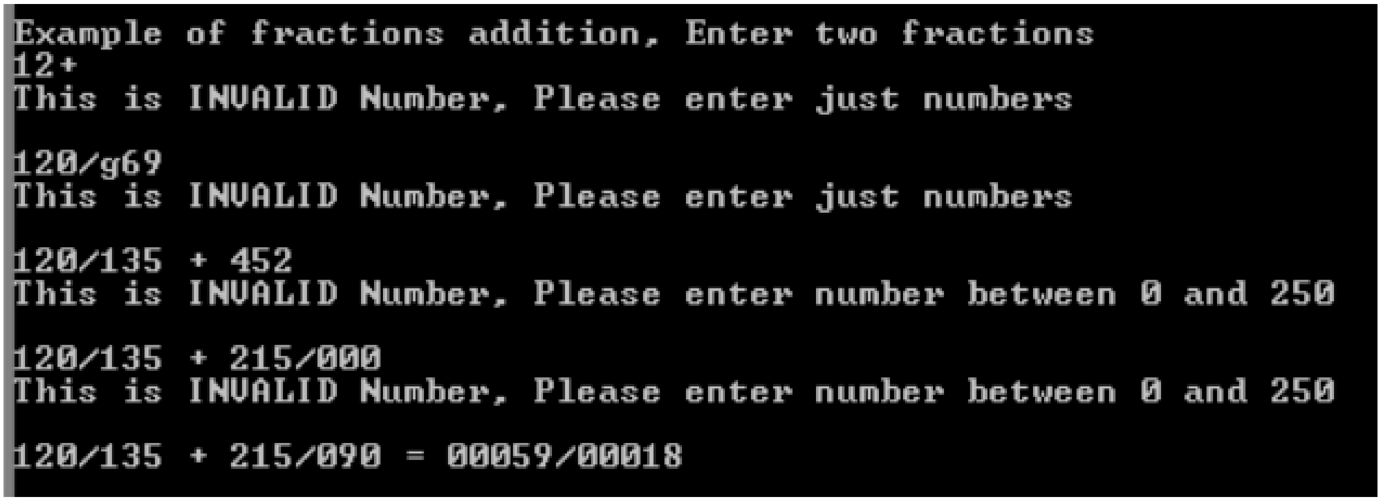
Figure 17: Sample of Part1 Outputs (With Error Messages)



```
Hello!


Please Enter Number 1: 240
Please Enter Number 2: 190



GCD = 00010
LCM = 04560
```

Figure 18: Sample of Part1 Outputs (Without Invalid Inputs)

Part2: Find the Result of Adding Two Fraction Using GCD

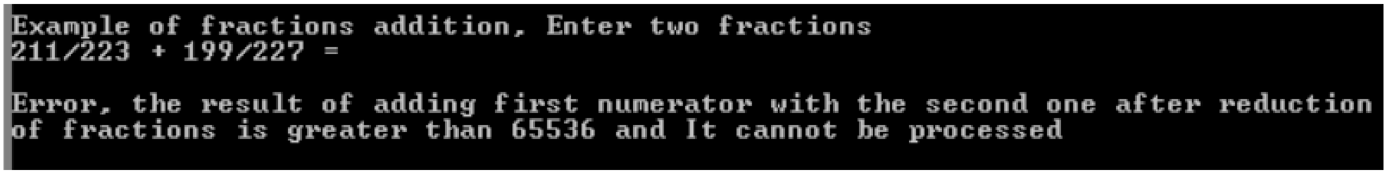


```
Example of fractions addition, Enter two fractions
12+
This is INVALID Number, Please enter just numbers

120/g69
This is INVALID Number, Please enter just numbers

120/135 + 452
This is INVALID Number, Please enter number between 0 and 250

120/135 + 215/000
This is INVALID Number, Please enter number between 0 and 250

120/135 + 215/090 = 00059/00018
```

Figure 19: 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).

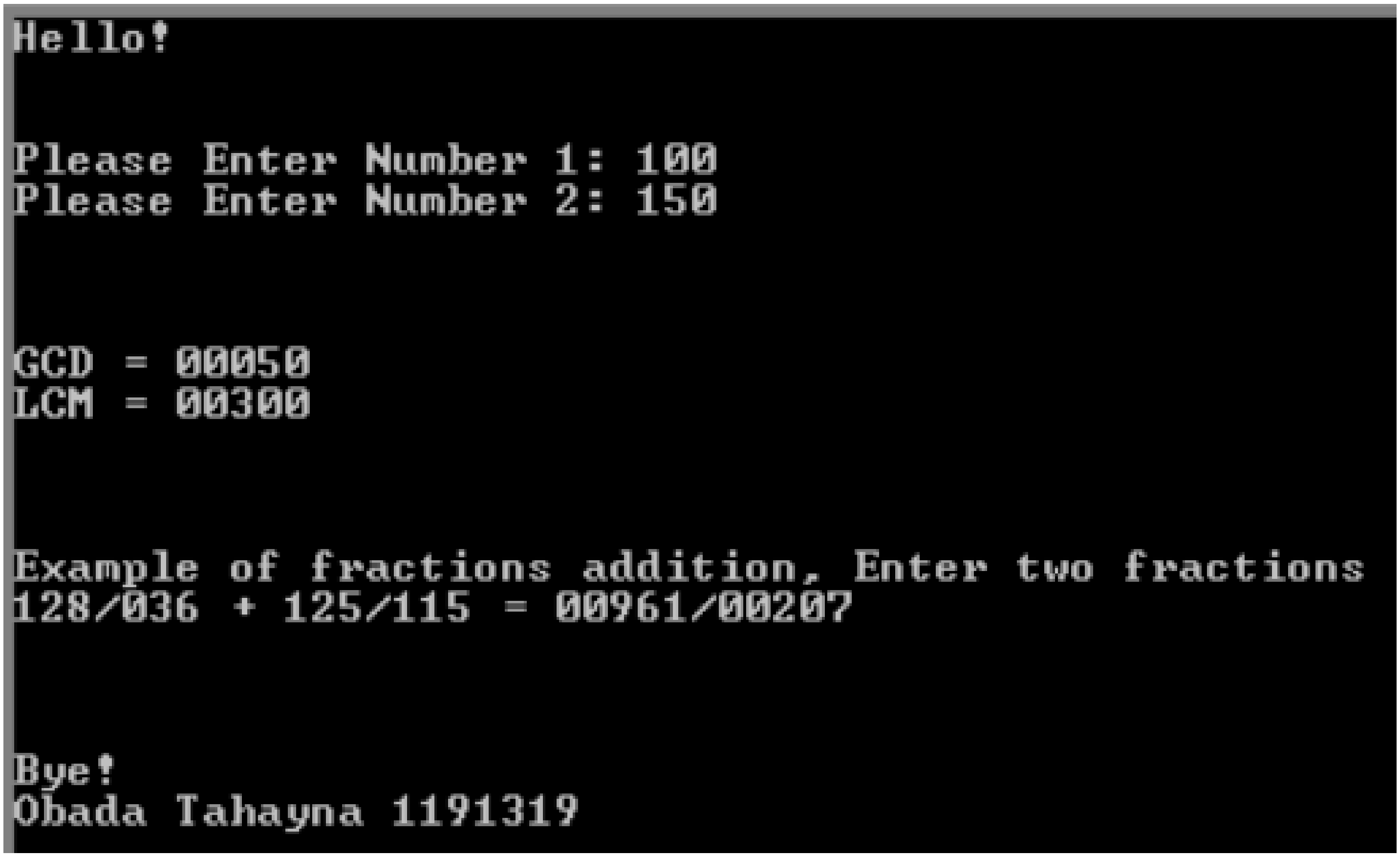
If the user enters numbers like 211/223 + 199/227, the two denominators are prime numbers, that means when we try to make the denominators same, it will be 47897/50621 + 50621/50621, and when we try to add them, the result will be 98518/50621, but the problem is the numerator cannot store in 16-bit register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



```
Example of fractions addition, Enter two fractions
211/223 + 199/227 =

Error, the result of adding first numerator with the second one after reduction
of fractions is greater than 65536 and It cannot be processed
```

Figure 20: Sample of Part2 Outputs (With Big Number Error)

Full Output



```
Hello!


Please Enter Number 1: 100
Please Enter Number 2: 150



GCD = 00050
LCM = 00300



Example of fractions addition, Enter two fractions
128/036 + 125/115 = 00961/00207



Bye!
Obada Tahayna 1191319
```

Figure 21: Full Output of The Program