California State University – Fullerton

College of Engineering and Computer Science

Equations Handler – Final Project

Gustavo Couto Vanin – 885517276

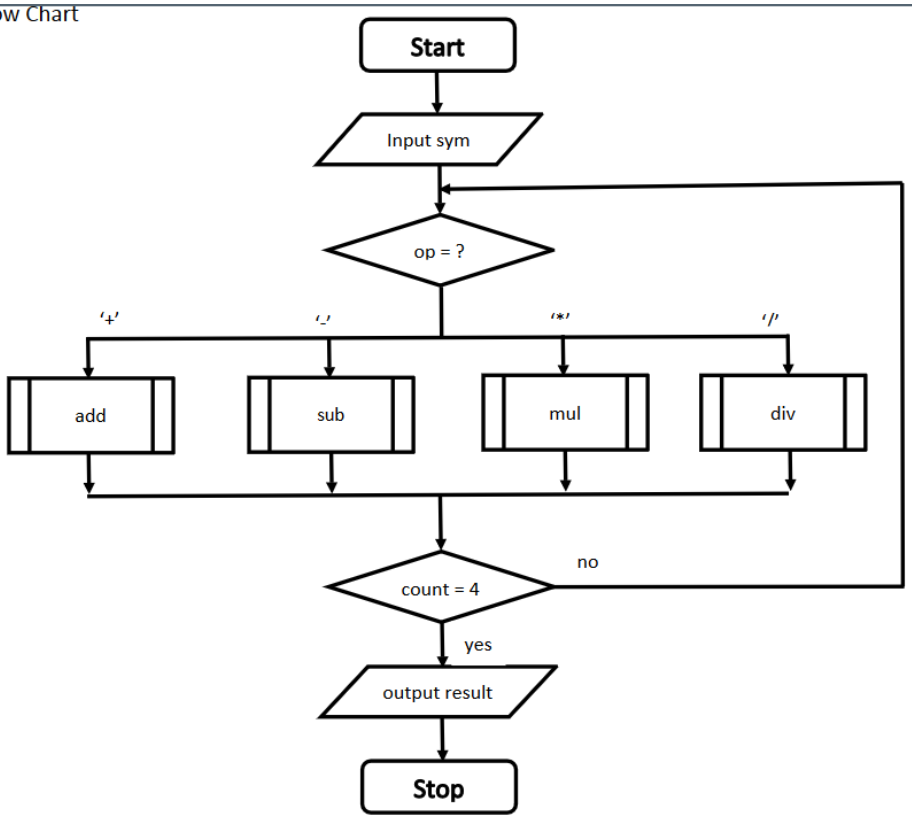
14th May, 2023

Introduction:

This project is a part of the final project for the CPSC 240: Computer Organization and Assembly Language (Spring 2023). In this project we were prompted to develop a program that takes in an equation with single integers only, that disregards the rules of an equation by performing a simple left to right operation. In this program, there was an assumption that the user input would not be negative, not have spaces in between the operations, and use the standard signs for the operations. Assembly has proven to be quite challenging for this project for a few reasons: ASCII code to decimal numbers, the other way around, and the operator handle. In order to transform ASCII code to decimal numbers, we had to mask off the first four bits of the hex code or simply subtract 0x30h from the hex value; that is due to the constant difference between decimal numbers and ASCII values. The other way around is basically the opposite, instead of subtracting, we add 0x30h from the hex value. Now, the operations were the trick part. Addition and Subtraction are quite simple and follow logic, multiplication and division though are quite different; they use pair keys, which is basically the decimal numbers of a decimal values. And those values are an offset of the most significant number.

Design Principle:

Flow Chart




```
%macro print 2                    ;Macro for Printing the Output
mov rax, 1
mov rdi, 1
mov rsi, %1
mov rdx, %2
syscall
%endmacro

%macro scan 2                     ;Macro For Scanning the User Input
mov rax, 0
mov rdi, 0
mov rsi, %1
```

```
mov rdx, %2
syscall
%endmacro

section .data
   msg1 db 'Enter Operations String: ',0
   msg2 db 'Result: ',0
   eq_buffer times 10 db 0
   result db 0

section .text
   global _start

_start:
   print msg1, 25                ;prompt the user
   scan eq_buffer, 10            ;intake the equation

   mov rsi, 0
   mov al, byte [eq_buffer+rsi]
   sub al, '0'
   mov byte[result], al    ;Initial Values for the result
   inc rsi
eqLoop:

   ;Check for operands match

   xor eax, eax
   mov bl, byte [eq_buffer+rsi+1]
   sub bl, '0'
   mov cl, byte [eq_buffer+rsi]
```

```
    cmp cl, '+'
    je add_eq

    cmp cl, '-'
    je sub_eq

    cmp cl, '*'
    je mul_eq

    cmp cl, '/'
    je div_eq

    cmp bl, 0
    jmp print_result

    inc rsi
    jmp eqLoop


    ;Operations for the Equations

add_eq:
    mov al, byte[result]
    add al, bl
    mov byte[result], al
    inc rsi
    jmp eqLoop

sub_eq:
```

```
    mov al, byte[result]
    sub al, bl
    sbb ah, 0
    mov byte[result], al
    inc rsi
    jmp eqLoop

mul_eq:
    mov al, byte[result]
    mul bl
    mov byte[result], al
    inc rsi
    jmp eqLoop

div_eq:
    mov al, byte[result]
    xor edx, edx
    div bl
    mov byte[result], al
    mov byte[result+1], ah
    inc rsi
    jmp eqLoop

print_result:

    ;print results

    print msg2, 8
```

```
    mov al, byte[result]
    add al, '0'
    mov byte[result], al
    print result, 100

exit_prog:
    mov rax, 1
    xor rbx, rbx
    int 0x80
```

All the code above is my developed final product and commented all through. The conversion is straight forward, subtract the hex value ‘0x30h’ to get the decimal value; and sum it to get the ASCII from decimal. The algorithm is as follows:

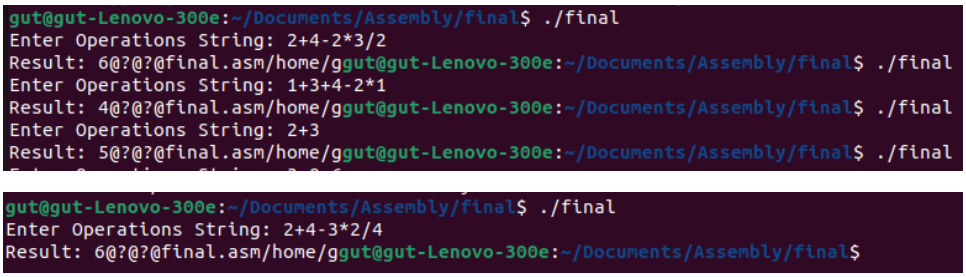
ASCII TO NUM

```
mov rax, 0 ;clear rax
mov rdi, 10 ;rdi = 10
mov rsi, 0 ;counter = rsi = 0
next0:
mov cl, byte[rbx+rsi] ;cl = [buffer+rsi]
and cl, 0fh ;convert ascii to number
add al, cl ;al = number
adc ah, 0 ;ah = 0
cmp rsi, 2 ;compare rsi with 2
je skip0 ;if rsi=2 goto skip0
mul di ;dx:ax = ax * di
skip0:
inc rsi ;rcx++
cmp rsi, 3 ;compare rcx with 3
jl next0 ;if rcx<3 goto next0
mov word[num], ax ;num = ax
```

NUM TO ASCII

```
mov ax, di
mov bx, 10 ;bx = 10
next2:
mov dx, 0 ;dx = 0
div bx ;dx=(dx:ax)%10, ax=(dx:ax)/10
```

```
add byte[ascii+rcx], dl ;ascii+0 = al + 30h
dec cx ;cx--
cmp cx, 0 ;compare cx and 0
jge next2 ;if (cx>=0) jump to next2
```



I’m extremely satisfied with the final product. The outputs were correct although very badly formatted. There were a few cases where the program was not working, most of them were when the results were negative and/or ending in decimal numbers. Overall, the project was successful taking into consideration the that we weren’t supposed to handle user input errors.

In conclusion, this project has taught me ASCII to Decimal code, as well as the opposite. And handling multiple operations and operators at the same time. Programming in assembly is more stressful than other languages, but it’s good to understand what is going on at the back end of the back end.