California State University, Fullerton

Computer Science Department

CPSC 240 Final Project

Kush Patel

885857847

11/27/2023

1. Introduction

There is a huge difference between keyboard input, which is known as ASCII code and decimal number, the ASCII code are keyboard input, in which each character on the keyboard has a number, even the non number characters, the number characters on the keyboard doesn't have a number that it is equal to itself, for instance the character of ‘7’ on the keyboard input is in reality is 55, while decimal numbers are the traditional numbers we use in life, the decimal number of ascii character ‘7’ is 55. The monitor out or the ASCII code is what it shows on a computer, for instance, if it displays 41, it's not the actual number 41, the actual number 41 is a decimal number, the ascii chart shows the number 41 and its decimal equivalent. The decimal numbers represent the number, whereas the ascii numbers or the output represents the output value but has a different number. We can do four different basic operations in assembly language which are addition, subtraction, multiplication, and division. To perform addition we set a value using the ‘mov’ command and then add it using the ‘add’ command. To perform subtraction we set a value using the ‘mov’ command and then subtract it using the ‘sub’ command. To perform multiplication we set a value using the ‘mov’ command and then multiply it using the ‘mul’ command.To perform division we set a value using the ‘mov’ command and then divide it using the ‘div’ command.

2. Design Principle, Algorithm, and Code

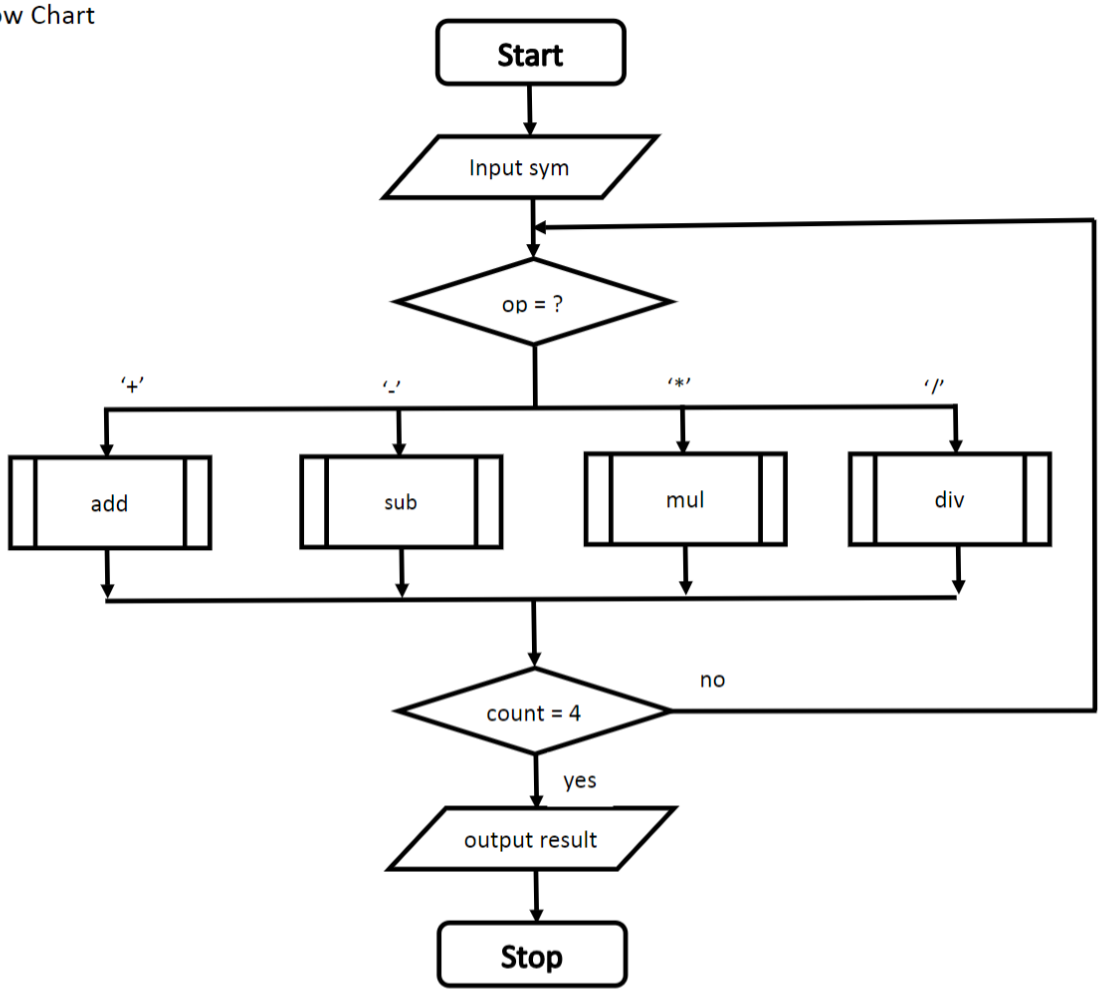
The entire assembly program for this calculator has a macro used for printing to the terminal. The entire program is a one-digit basic calculator which uses four basic and different operators which are the plus, minus, multiply, and divide operators. At the beginning of the program there are two macros, one for printing to the terminal or

keyboard and the other was for receiving input from the terminal or keyboard, each macro had a basic mov statements which if it was a ‘1’ then we were printing to the terminal and if it was ‘0’ we receive input from the terminal. We had our data values, which were the input, numbers, symbols, and total which were all located in the “section .bss” and used the data types ‘resb’ and they were all initialized to ‘1’, with buffer being initialized to ‘10’. In the “section .data”, there was a output string where it would ask you for the input, the string that asks for the input says “Enter Operation String”, which asks you for four numbers and four different math symbols, which the symbols can be in any order, like multiplication can go first or addition can go first, vise versa, the “ascii db “00”, 10”, will output the operating string result as a two digit string, like if the result ends up like a one digit number it’ll be outputted like “09”. Every assembly program must have a .txt section in which there must be a global “_whatevername”, in order for it to compile and run without errors. The “_whatevername” is where everything in the program runs, and “_whatevername” is called “_start”, The section below the print msg1 and scan intakes the values input by the user and converts them into numerical values, that’ll be later calculated. There's a command that says “mov al, byte[buffer + 0]”, which converts the first input ascii value to a numerical value. The command “mov byte[num1], al”, stores the first character into the num1 variable. There's a command that says “mov al, byte[buffer + 2]”, which converts the second input ascii value to a numerical value. The command “mov byte[num2], al”, stores the second character into the num2 variable. There's a command that says “mov al, byte[buffer + 3]”, which converts the third input ascii value to a numerical value. The command “mov byte[num3], al”, stores the third character into the num3 variable. There's a command

that says “mov al, byte[buffer + 4]”, which converts the fourth input ascii value to a numerical value. The command “mov byte[num4], al”, stores the fourth character into the num4 variable. There's a command that says “mov al, byte[buffer + 5]”, which converts the fifth input ascii value to a numerical value. The command “mov byte[num5], al”, stores the fifth character into the num5 variable. There's a command that says “mov al, byte[buffer + 6]”, which converts the sixth input ascii value to a numerical value. The command “mov byte[num6], al”, stores the sixth character into the num6 variable. There's a command that says “mov al, byte[buffer + 7]”, which converts the seventh input ascii value to a numerical value. The command “mov byte[num7], al”, stores the seventh character into the num7 variable. There's a command that says “mov al, byte[buffer + 8]”, which converts the eighth input ascii value to a numerical value. The command “mov byte[num8], al”, stores the eight characters into the num8 variable. There’s a command called “mov al, byte[num1]” and mov byte[total], al, which stores num1 as the total value. There is section were it evaluates the operator sym1 whether if it is “+”, “-”, “/”, or “*” and jumps to the next instruction. When it calls the cmp command it compares sym1 to “+” and if sym1 does not equal “+”, it jumps to “skip11” section, and skips everything else in that section. The commands “mov dil, byte[total]” and “mov sil, byte[num2]”, which moves total and num2 into dil and sil and continues to do the operation. There is a “call addition” command which if sym1 equals “+”, it calls the addition function and after that it jumps to the second section. Then it goes through a skip11 section which does everything as the previous section except it only compares the “-” operator and calls the subtraction function and if there is no subtraction symbol for sym1, then it jumps to skip12, which then compares the multiplication symbol and

performs the multiplication function and after it jumps to the second section. If there is no “*” symbol for sym1, it then lastly jumps to skip13, which then compares the division to check to see if there is a “/.” symbol, if there is then it calls the division symbol, and regardless if there is a “/” symbol or not it jumps to the second section, which performs all the same instructions and commands as the first section except it's only checking for the second symbol and third number, it then jumps to the third section after and does the same thing as the first and second section except it's only checking for the third symbol and fourth number, it final jumps to the fourth and last section and does the same thing as the first, second, and third section except it's only checking for the fourth symbol and fifth number and after all that it jumps to done, where it gets the total result and prints and outputs the total result to the terminal. The program ends with instructions and commands “mov rax,60”, “mov ‘rdi’, 0”, and “syscall”. Outside the program there are four functions written, which are “addition”, “subtraction”, “multiplication”, and “division”. The addition function calls the add instructions and updates the value of total and has the “ret” command to return back to the line prior to calling the function. The subtraction function calls the sub instructions and updates the value of total and has the “ret” command to return back to the line prior to calling the function. The multiplication function calls the ‘mul’ instructions and updates the value of total and has the “ret” command to return back to the line prior to calling the function. The division function calls the ‘div’ instructions and updates the value of total and has the “ret” command to return back to the line prior to calling the function.

Flow Chart



3. Simulation Result

The simulation results require you to input a math function as a string and it will loop through the string to check what numbers and symbols are there to perform the calculation. If the operation string is “4+4-4*2/4”, then it results in 02, because it doesn't care about “PEMDAS”, it checks which numbers and symbols are first. You can

implement and input the same symbol as well. All the symbols don't have to be different. If I input "4+4+4+4+4", I will get an answer. If I input "3+3", the program will crash because they want the full operation string. There's a loop that will run through the operation string to check the symbols and numbers and if the symbol isn't there or the number, it'll crash.

```
Enter Operations String: 2+3-2*2/1
06
kushpatelj86@DESKTOP-T9949G2:/mnt/c/Users/Kush/myworkspace/CPSC 240/final$ ./final
Enter Operations String: 2+3+3+3
11
```

```
Enter Operations String: 20-10-10-10
^C
kushpatelj86@DESKTOP-T9949G2:/mnt/c/Users/Kush/myworkspace/CPSC 240/final$ ./final
Enter Operations String: 100-10-10-10
00
kushpatelj86@DESKTOP-T9949G2:/mnt/c/Users/Kush/myworkspace/CPSC 240/final$ 10
10: command not found
kushpatelj86@DESKTOP-T9949G2:/mnt/c/Users/Kush/myworkspace/CPSC 240/final$ ./final
Enter Operations String: 10*10*10*10
^C
kushpatelj86@DESKTOP-T9949G2:/mnt/c/Users/Kush/myworkspace/CPSC 240/final$ ./final
Enter Operations String: 1*2*2*5
20
kushpatelj86@DESKTOP-T9949G2:/mnt/c/Users/Kush/myworkspace/CPSC 240/final$ ./final
Enter Operations String: 1/2/3/4
00
kushpatelj86@DESKTOP-T9949G2:/mnt/c/Users/Kush/myworkspace/CPSC 240/final$ ./final
Enter Operations String: 10/8/6/4
^C
kushpatelj86@DESKTOP-T9949G2:/mnt/c/Users/Kush/myworkspace/CPSC 240/final$ ./final
Enter Operations String: 9/2/2/2
01
kushpatelj86@DESKTOP-T9949G2:/mnt/c/Users/Kush/myworkspace/CPSC 240/final$
```

4. Conclusion

In conclusion, the calculator program was written in assembly and was strict about numbers and symbols. The program ran without errors. The Calculator had to use loops and if statements inside the loop. ASCII and regular numbers are different because ASCII has regular numbers, but in characters and the regular number value of those are higher than they should be. The calculator has only basic commands which are addition, subtraction, multiplication, and division. We had to call each command using a function, while looping through a string, we can include any symbol in any order.