ECE 3210: Microprocessor Engineering
Laboratory

Lab # 4

Lab Title Calculator

Group# 9

Group Names: Olasumbo Babalola, Mason Adamovicz

Teaching Assistant Use Only:

Points Earned	Reasons for Deduction
Pre-lab:	
Post Lab report:	
Demonstration:	
Final Lab Grade:	
Comments to students:	

Objective:

The objective of this lab is to be able to create a program that runs efficiently while calculating different algebraic expression. At the end of this lab, we will be able to use the different techniques like MACROS to make our program readable, use stacks and finally the use of instructions to perform arithmetic/logic operations, and comparisons.

Problems Encountered:

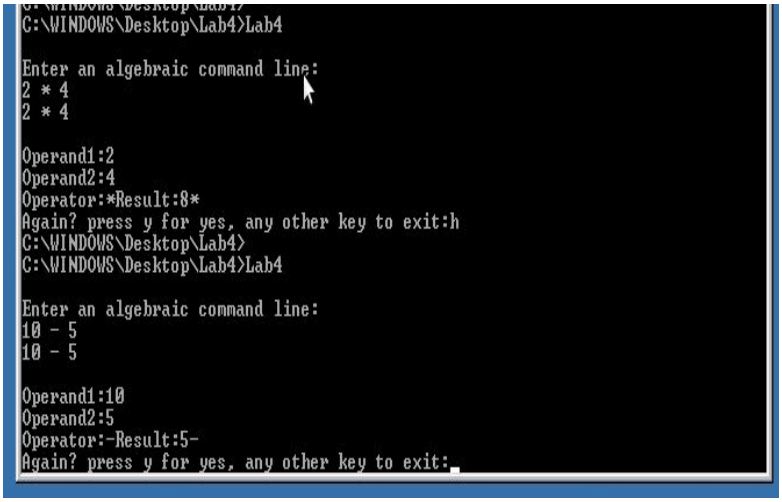
One of the problems we encountered was we lost all our code from the first phase of the Lab so we had to start from scratch during the next lab class. At first we had difficulties with storing the input string the right way. We were trying to add 12 + 34 but for some odd reason we were getting 12 + 43. We were able to fix this problem by going back into our code and changing it up. While creating a procedure from ASCII to HEX we had difficulties with using a random register to represent N1_H or N2_H. We were able to figure this out and after doing s we could call ASCII to Hex in the main function for the two different numbers without having any issues. We also had to move our different jump statements around because we kept getting an out of range error

Conclusion stating what you have learned:

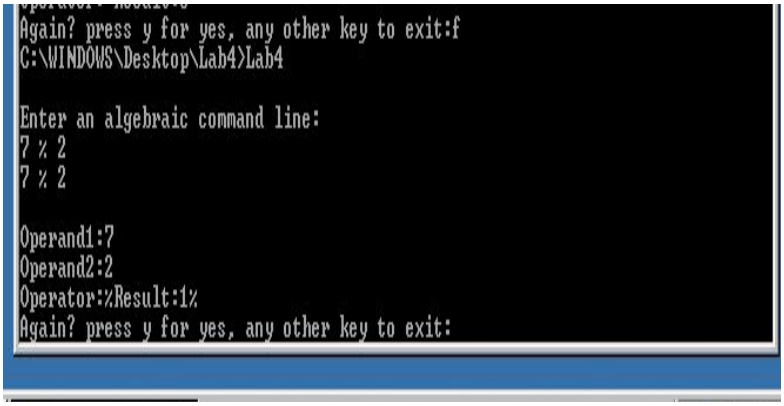
This lab helped us understand how to use different arithmetic functions, and how to change ASCII values into Hex values and vise verse. We've also learned the hard way that we need to back up our saved data onto an external device, as we lost our first week's worth of code due to the VIM machine crashing. We have also learned that the JMP commands need to be within range in our code, otherwise we can implement multiple JMP statements to leapfrog to where the destination of the jump is.

Sample testing scenarios; negative numbers, divide by zero, multiply two big numbers ...etc. Provide screenshots of the results.

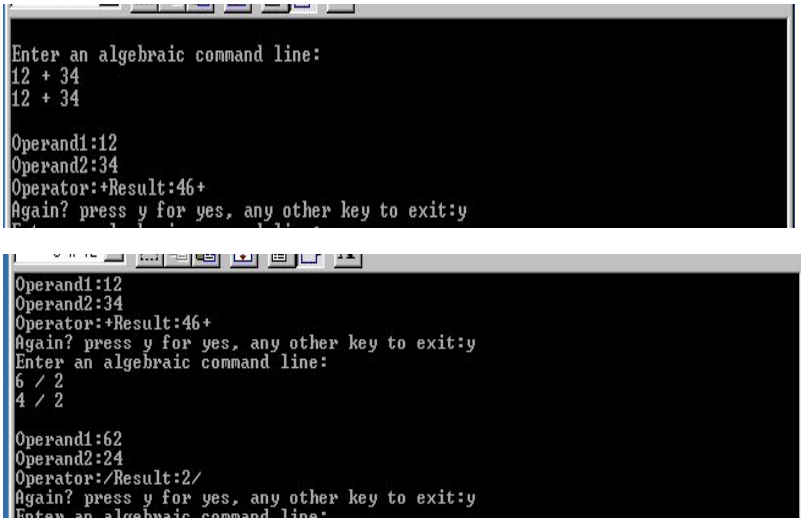
Multiplication and Subtraction



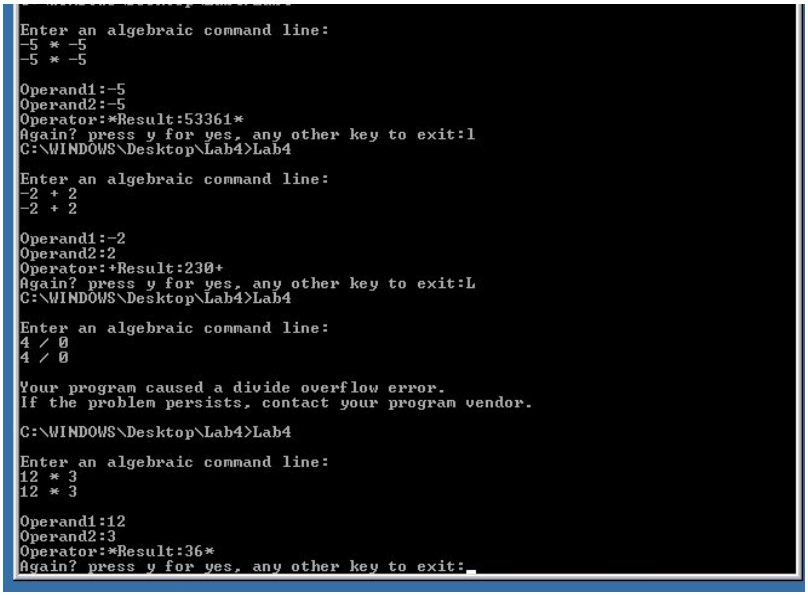
Modulus



Addition and Division

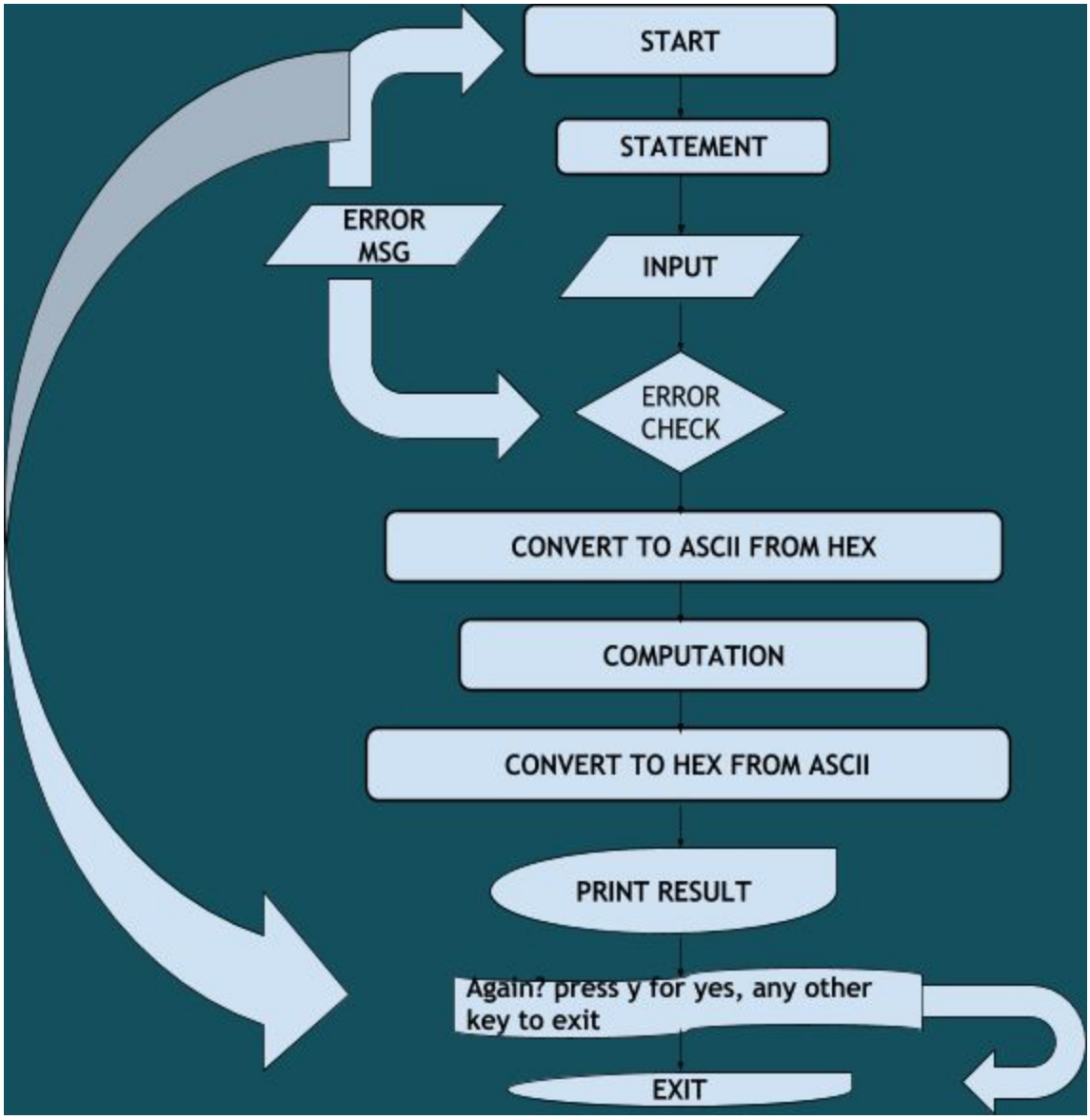


Division by 0 / Negative number and Huge number Multiplication



Explain your approach to solve the problem:

We started by getting the operation values and the operand from the user. Then we make sure that the user inputted the arithmetic equation correctly if not, we outputted an error message that implicitly tells the user how to type in the equation correctly. After the correct arithmetic is entered we proceed by changing the Ascii equation into hexadecimal number and once we have the hexadecimal number we do all the different computation and stored them in a register. After computation is done, we convert from hex to Ascii then print it back to the user. Finally, we ask the user if they want to do another computation or not, then proceed again.



Any special instructions used, and why? Hint: PUSH /POP /LODSB, STOSB, NOT, AND, TEST ...etc

We used

PUSH/POP - We used Push during our Hex to ASCII conversion. We used it to push in remainder numbers, which when POP backed up will give s our ASCII number.

LODSB- We used LODSB to load Al, AX with data stored in SI.

CLD - We used this to clear our register in order to prevent any error.

IMUL/IDIV - We used this to multiply or divide both positive and negative number.

Known problems in your code, any special case that would cause your code to fail (if any), and why?

Using larger numbers for multiplication or division can yield incorrect values due to them being bigger than the registers, dividing by 0 will produce an error and crash the program due to us not putting in a check for when dividing by zero. Negative numbers also will not work, as we were pressed for time and didn't get to put the code in to check for negative numbers. And selecting to restart the calculator at the end of a calculation will end up giving the wrong values for any calculation due to the registers not being wiped every time.

Source code files, well commented and indented.

```
;LAB 4 PROGRAM

;FILENAME: lab4.asm

;FILE TYPE: EXE

.MODEL MEDIUM

.STACK 1024

.DATA

    Buffer   DB 20

    NUM      DB ?

    ACT_BUF  DB 20 DUP('$')

    MSG      DB 0DH,0AH,'Enter an algebraic command line:',0DH,0AH,'$'

    NEWLINE  DB 0DH,0AH,'$'

    ERROR    DB 'Error, invalid input',0DH,0AH,'Input format:
Operand1 Operator Operand2',0DH,0AH,'Operand: decimal
numbers',0DH,0AH,'Operator: + - * / %$',0DH,0AH

    AGAIN    DB 'Again? press y for yes, any other key to exit:$'

    OPERAND1 DB 'Operand1:$'
```

```
    OPERAND2 DB 'Operand2:$'

    OPERATOR DB 'Operator:$'

    RESULTS  DB 'Result:$'

    N1_A  DB 4 DUP ('$')

    ;N1_A      DB ?

    N2_A   DB 4 DUP ('$')

    ;N2_A      DB ?

    N1_H      DB 0

    N2_H      DB 0

    OPT       DB ?

    Result_H  DW 0

    Result_A  DB 20 DUP ('$')

    choice    DB 0

.CODE

    .STARTUP

MAIN PROC FAR

START:    MOV AX,@DATA          ;startup
        MOV DS,AX
        MOV ES,AX

        LEA DX,MSG            ;display prompt message
        MOV AH,09h
        INT 21h


        ;Enter String
        MOV  AH,0AH
```

```
        MOV  DX,OFFSET Buffer
        INT 21H

        ;Display  new line
        MOV DX, OFFSET NEWLINE
        MOV AH,09H
        INT 21H

        ;Display the entered string
        MOV DX, OFFSET ACT_BUF
        MOV AH, 09H
        INT 21H

        ; Display  new line
        MOV DX, OFFSET NEWLINE
        MOV AH,09H
        INT 21H

        LEA DI, N1_A
        LEA SI, Act_Buf

        MOV CX, 0000H
        MOV CL, NUM

T:      CLD
        LODSB
        CMP AL, 20H
        JNE E                   ;error check jump
        JE T2
        CMP AL, 0DH
        JNE E                   ;error check jump
        JE T3
        MOV [DI], AL
        INC DI
        LOOP T

T2:     LODSB
        MOV OPT, AL
        ADD SI, 1
        LEA DI, N2_A
        SUB CX, 1
        LOOP T
```

```
T3:     MOV SI, OFFSET N1_A        ;ASCII to Hex conversion
        CALL A2H
        MOV N1_H, BL


        MOV SI, OFFSET N2_A
        CALL A2H
        MOV N2_H, BL
        MOV AX, 0000H
        MOV AL, N1_H
        MOV BX, 0000H
        MOV BL, N2_H

        JMP OPERATE

E:      MOV DX, OFFSET ERROR
        MOV AH,09H
        INT 21H
        JMP OPTION

OPERATE:  ;COMPUTATIONS
        CMP OPT, '+'
        JE PLUS

        CMP OPT, '-'
        JE MINUS

        CMP OPT, '*'
        JE MULT

        CMP OPT, '/'
        JE QUOTIENT

        CMP OPT, '%'
        JE MODULUS

PLUS:   ADD AX, BX
        MOV Result_H, AX
        JMP HEX

MINUS:  SUB AX, BX
        MOV Result_H, AX
        JMP HEX
```

```
MULT:     IMUL BX
          MOV Result_H,AX
          JMP HEX

QUOTIENT: MOV DX, 0000H
          IDIV BX
          MOV Result_H, AX
          JMP HEX

MODULUS:  MOV DX, 0000H
          IDIV BX
          MOV Result_H, DX
          JMP HEX

START1:   JMP START

HEX:      MOV SI, OFFSET Result_H          ;call hex 2 ascii proc
          CALL H2A

          MOV DX, OFFSET NEWLINE
          MOV AH,09H
          INT 21H

          MOV DX, OFFSET OPERAND1
          MOV AH,09H
          INT 21H

          MOV DL, N1_A
          MOV AH, 02H
          INT 21H

          MOV DX, OFFSET NEWLINE
          MOV AH,09H
          INT 21H

          MOV DX, OFFSET OPERAND2
          MOV AH,09H
          INT 21H

          MOV DL, N2_A
          MOV AH, 02H
          INT 21H
```

```
        MOV DX, OFFSET NEWLINE
        MOV AH,09H
        INT 21H

        MOV DX, OFFSET OPERATOR
        MOV AH,09H
        INT 21H

        MOV DL, OPT
        MOV AH, 02H
        INT 21H

        MOV DX, OFFSET RESULTS
        MOV AH,09H
        INT 21H

        MOV DX, OFFSET Result_A
        MOV AH,09H
        INT 21H

        MOV DL, OPT
        MOV AH, 02H
        INT 21H

OPTION:   MOV DX, OFFSET NEWLINE
        MOV AH,09H
        INT 21H

        MOV DX, OFFSET AGAIN
        MOV AH,09H
        INT 21H

        MOV AH, 1
        INT 21H
        MOV choice, AL

        CMP AL, 'y'
        JE  START1
   .EXIT
MAIN ENDP

A2H PROC NEAR
```

```
        MOV BL, 00H
BEGIN:    MOV CL, [SI]
        CMP CL, '$'
        JE  exit
        SUB CL, 30H
        MOV AX, 10
        MUL BX
        ADD AX, CX
        MOV BX, AX
        INC SI
        JMP BEGIN
exit:
   RET
A2H ENDP

H2A PROC NEAR
        LEA BX, Result_A
        Mov CX, 10
        Push CX
        Mov AX, Result_H

L1:       Mov DX, 0      ;Diving ASCII Number
        DIV CX
        PUSH DX
        CMP AX, 0
        JNZ L1

L2:       POP DX
        CMP DX, CX
        JE L4
        ADD DL, 30H

L3:       MOV [BX], DL
        INC BX
        JMP L2

L4:       MOV byte ptr[BX], '$'


RET
H2A ENDP
END
```