HOMEWORK 1 REPORT

CmpE 230

Bekir Yıldırım - 201400054
Alkım Ece Toprak - 2017400294

- **BASICS**

We started our implementation with a basic version of the project. At first, implementing various trial cases helped us grasp the main idea behind the 8086 Assembly language. By trial and error; we learnt how to turn the for and while loops, conditional statements (if, else if, else) we know from other high-level languages into assembly language using compare, jump, loop counter instructions.
At first, we worked with single digit inputs and implemented codes that performed a single type of operation. We did not want to mix things up with jump statements and labels for different operations.
After understanding how things work, we added different features one by one, testing and debugging in every step.

- **IMPROVING**

After obtaining the main outline of the project, we started dealing with the input format. The postfix notation required us to operate on the latest 2 numbers that have been calculated. To implement this, we used stacks. The input we get can be long or short therefore creating different variables and storing the input values in them would be meaningless since we do not know how many numbers we get in the testcases. So, we started pushing the values to the stack and popping them whenever a calculation character is seen in the input string.
After operative characters are seen in the input, we added different parts (labels) that act according to the character's task. Since there are built-in instructions for the operations, these are quite simple labels.
We also had to change the way we take the input values from the user because the input could contain up to 16-bit long values. To make this happen, we implemented the "readInput" label:

```
readInput:
        mov bl, al          ; bl stores the most recent input character
        mov bh, 00h
        mov ah, 00          ; ax stores 10h (16 in decimal)
```

```
        mov ax, 10h
        mul cx          ; ax=cx*10h
                                ; cx contains the value of the input number before the latest
                                ; digit “bl” is written
        mov cx, ax              ; cx becomes ax
        add cx, bx              ; cx=cx+bx
        jmp digitLoop
```

For this part of the code, let us consider the input string “3 2A4 +”. When “3” is seen, it is written to the variable cx. After seeing the “Space” character after “3”, cx is pushed to the stack. To calculate and store “2A49” in stack, we move to the given part of the code. Initially 2 is stored in cx. When “A” is written immediately after “2”, cx must become “2A” which is equal to 2*16^1+10*16^0.
So, cx is multiplied with 16 and becomes 2*16^1 and “A” (or 10 in decimal) is added to cx. Same thing is true for “2A4”. 2A4 = 2*16^2+10*16^1+4*16^0 which is also equal to 16*(“2A”) +4.

- **PROBLEMS**

Since we started with single digit values, printing part of the code was a problem for us. We could not figure out how to print results that are bigger than 4 bits. We had the final result pushed to the stack, waiting to be printed. We then found that the rightmost digit can be written by taking the remainder when the result is divided to 16. So "result modulo 16" gave us the last digit. We used the division instruction "div" that writes the remainder to "dx" for this and created this part of the code:

```
exit:
        mov ax, cx              ; ax contains the final result
        mov cx, 0004h; cx is 4 since 16-bit values correspond to hex
                                ; values with 4 digits
outputLoop:
        mov bx, 10h
        div bx
        PUSH dx                 ; dx now has the last uncalculated digit of the
                                ; answer (dx=answer%16)
        dec cx          ; cx is decreased until all digits(dx) are pushed
        cmp cx, 0000h; when cx equals 0, every digit of the result
                                ; is pushed to stack
        jz final        ; if cx=0 jump to the "final" label
                        jmp outputLoop          ; else, cx is not equal to 0, continue pushing
                                ; digits to stack and decreasing cx
```

This code segment popped the final result from stack and pushed the digits back starting from the rightmost one.
We were now able to print all digits by popping everything from stack. But there was another problem. Since we stored our result in a 16-bit long variable and didn't check the size of the result after computation, results that we printed were also 16-bit long. So instead of "13B", our program would print "013B".

To solve this, we added a couple of labels and changed the code a bit:

```
exit:                           ; the latest result calculated before the "enter" input is copied to ax
        mov ax, cx
        PUSH cx                 ; and pushed to the stack
        mov cx, 0001h
                                ; stack now contains the final result only
exitLoop:
        mov bx, 10h
        div bx          ; the answer (ax) is divided to 16 and the last digit
                        ; (remainder of the division) is stored in dx
        mov dx, 0000h
        cmp ax, 0000h; if ax is not equal to 0, jump to countCX
        jnz countCX
        jmp output              ; else jump to output
                                ; if ax is equal to 0, it means that the whole answer
                        ; is pushed to stack
countCX:
        inc cx          ; cx is our loop counter for exitLoop
                                ; when the loop ends, cx equals to the number of
                        ; digits in our result
        jmp exitLoop
output:
        POP ax          ; ax contains the final result
        mov var1, cx    ; var1 contains the number of digits of the result
outputLoop:
        mov bx, 10h
        div bx
        PUSH dx                 ; dx has the last uncalculated digit of the answer
                        ; (dx=answer%16)
        dec cx          ; cx is decreased until all of the digits(dx) are
                        ; pushed to the stack
        cmp cx, 0000h; when cx equals 0, every digit of the final result
                        ; is pushed to the stack
        jz final        ; if cx=0 jump to the "final" label
        jmp outputLoop          ; else if cx is not equal to 0, continue pushing
                        ; digits to the stack and decreasing cx
```

In this part, we divide the result to 16 until the result part (ax) becomes 0 and store the size of the result with loop counter cx. Var1 now contains the number of digits of our result. So in the "output" label, var1 is used instead of 0004h.