CSUS

SCHOOL OF ENGINEERING AND COMPUTER SCIENCE

Department of Computer Science

CSC 35

Spring 2021

Dr. Ghansah

**Lab #7: Input/Output, Multiple Digits, Parameter Passing, Conditional Branch**

**\*\*\*\*NOTE CHANGE THE FORMULA FOR W**

**Purpose:** The main purpose of this lab assignment is to demonstrate understanding of how to use procedures (subroutines) in a *modular* fashion and *parameter passing* mechanisms. Another objective is to learn how to handle multiple decimal digits that have to be collected from the keyboard without leading zeros as well as multiple decimal digits output to the screen. It requires modification of the previous lab.

**Introduction:** In previous labs you learned how to output a single ASCII decimal value to the screen using the Irvine Library procedure *Writechar*. In lab6 you learned how to input multiple ASCII decimal digits from the keyboard using a procedure. Procedures typically are passed parameters (if needed) before they are called. The output of the procedure can also be a parameter if necessary. In Intel Assembly, parameters can be passed via registers or the stack. In this assignment you can use either method to pass parameters. Parameter passing by Registers is easier and is recommended.

**Requirements:** Since lab 6 is an extension of previous labs,modify Labs 6 as follows

1. **Multiple Decimal Digit Output**: Use the algorithm for outputting multiple decimal digits to the screen. It involves converting a binary number to a variable number of decimal digits. This is done by repeated division of quotients by 10 and keeping the remainders on a stack. The remainders in the reverse order of digits represent the decimal number. A stack can be used for storing and printing remainders because the latter can be retrieved from the stack in reverse order. As shown below the formula for computing output number W has been modified in such a way that it can produce one or more decimal digits depending on values of X and Y read from the keyboard. Here is the algorithm
2. Set a counter to zero.
3. Get number into the dividend registers for proper size.
4. Divide the dividend by 10 (use 16 bit divisor ).
5. Add ASCII value of 0x30 (30h) to the remainder digit. This is how to convert to ASCII
6. Save remainder digit on stack. (use PUSH instruction)
7. Add to counter of digits saved on stack
8. Zero out remainder register.
9. Test the quotient (next dividend) to see if its zero.
10. If No then go to step 3
11. If Yes print the digits from the stack one by one. (Use POP instruction and WriteNum procedure which in turn uses Irvine library call WriteChar to print each number to the screen.)
12. **Variable Number of Input Decimal Digits without leading zeroes**: Implement complete algorithm for inputting arbitrary number of decimal digits from the keyboard such that exact number of decimal digits is printed. That is, no leading zeros. In this case the program will check for Carriage Return ASCII character *0dh* which is the character obtained from the keyboard when you press the ENTER key. Thus 0040 will be entered from the keyboard as 40 ENTER. This means that unlike the previous lab, you should avoid using the LOOP instruction.
13. **Modularity**: We want to see better modularity of procedures via proper parameter passing. You will find it easier to use registers to pass parameters. No procedure should access parameters via global variables. Parameters can be passed by value or by reference as needed. Modularity also means that block of code that is used two or more times should be placed in a procedure.
14. **Procedures called in Main:** The main program will have calls to 3 major procedures similar to lab 6 but some of the procedures can call other procedures if the need arises. Thus, proper modularity might call for you to create more than 3 procedures.
15. **Main Loop**: The main Program stays in a loop until ‘q’ (quit) key is pressed.

**Procedure:** Write an assembly program that a) receives the variable size decimal numbers X and Y from the keyboard. Each number ends with the Enter Key (0dh); b) Then the program computes W (where computation steps are same as in previous lab up to the step before the last); c) Then it prints W on the screen and stays in a looped session until the key ‘q’ (quit) is pressed. The Step by Step Pseudocode of the main program is:

1. Get variable X from Keyboard. Ends with Enter Key (0dh)
2. Get variable Y from Keyboard. Ends with Enter Key (0dh)
3. Compute W
4. Output W
5. Prompt to continue or quit
6. If continue Go to step 1 else Exit

Specifics of the algorithm in C-like notation is as follows

Main()

{

int X, Y, W;

int A=90;

int B=60;

Do { *…*

*X=GetInput(XPrompt) ; input X from the Keyboard*

*Y=GetInput(YPrompt) ; input Y from the Keyboard*

W = CalculateW(X,Y) ;compute W

outW(); ; output W to screen

While (Readchar <> ‘q’)

}

Calculate (int m, int n)

{

int loc1, loc2, loc3, sum ;local variables

loc1 = n \* 160 + m \* 2

loc2 = A\* 950 + B\*350

loc3 = loc2 – loc1

n=3000

n=n-1

Sum = loc3/16 +n + n/4 + n/200

*Return ( sum % 947 + 345); sum* ***%*** *947 = sum* ***MOD*** *947*

*}*

outW(int W)

{

…

*Lines of code to output W potentially multidigit decimal to the screen.*See“Multiple Decimal Digit Output” *under requirements above for Algorithm*

…

}

**A typical session will look like the following**

Input X= ?

Input Y= ??

Calculating W

Output W= ???

Press q to quit or some other key to continue

a

Input X= ??

Input Y= ???

Calculating W

Output W= ???

Press q to quit or some other key to continue

a

Input X= ??????

Input Y= ???

Calculating W

Output W= ??

Press q to quit or some other key to continue

q

**Note1:** NO PROCEDURES ARE ALLOWED TO USE GLOBAL VARIABLES EXCEPT FOR VARIABLES FOR PROMPTING and variables A and B which are constants. CLEARLY, X, Y, AND W CANNOT BE ACCESSED BY PROCEDURES since they are passed as parameters. Additionally, LOC1, LOC2, LOC3, AND SUM CANNOT BE ASSESSED BY ANY PROCEDURE EITHER BUT THEY CAN BE REPLACED BY REGISTERS IN THE COMPUTATION OF W. An alternative is to use them as LOCAL variables within the CalculateW procedure.

**Note2**: You may not use any library functions other than Irvine Library functions *CrLf*, *ReadChar*, *WriteChar*, and *WriteString*

**Note3**: Your program should be able to handle **variable number of unsigned decimal digits** for X, Y and W.

**HINTS:**

**1)** Declare variables for strings in the data area. For instance

Add *continue* variable for prompt string as follows.

Continue BYTE “Press q to quit or some other key to continue”, 0

**2**) Main Procedure Outline in Assembly:

.data

…

.code

Main PROC

…

Top:

Mov si, offset InputX ;input parameter passing

Call GetInput

Mov X,ax ;output parameter passing

Mov si, offset InputY

Call GetInput

Mov Y,ax

Mov si,X ;input parameter passing

Mov di,Y ; input parameter passing

Call calculateW

Mov W, ax ;output parameter passing

Mov ax,W ;input parameter passing

Call OutW

…

Code for prompting to continue, collecting single key from keyboard, and branching to the ‘Top’ Label if key ‘q’ is not pressed

Main ENDP

**Testing:** Be sure to test your program and make sure it works before you submit it to your lab instructor on CANVAS as specified below.

**Demonstration**: Demonstrate your program to the instructor with screen shots similar to the session above. INSTRUCTOR WILL CHECK your source code 1) Whether parameters are passed before calls are made to procedures, 2) Single procedure for Multiple digit inputs, X and Y. 3) Whether global variables are avoided except for the exceptions of printing prompts are specified above, 4) Whether global variables are used in procedures illegally. 5) Whether your program handles multiple digit outputs in a procedure, 6) The lab instructor will also assemble and run the documented source code you upload to CANVAS as specified below. The instructor will check whether your program adheres to the session above as specified by your screen shots.

**Submission:** Submit electronic copy of your program to CANVAS including a well *Documented program (source code)* and output (screen shots). **Filenames must be according to the format specified in the syllabus**