CSUS

SCHOOL OF ENGINEERING AND COMPUTER SCIENCE

Department of Computer Science

CSC 35

Spring 2021

Dr. Ghansah

**Lab #9: Signed Arithmetic, Stack Frame**

**Purpose:** The main objective of this lab assignment is to demonstrate understanding of how to create and use procedures (subroutines) whose parameters are passed via the Stack. Another objective is to learn how to declare and use signed variables for handling arithmetic computations involving signed multiplication and division instructions. This lab is an extension of Lab 7. However, the main program for this assignment has been supplied. You will only create procedures.

**Introduction:**

**1)** In previous labs you learned how to pass procedure input parameters via registers which is faster but less flexible. In this lab you are going to pass input parameters via the stack which is slower but more flexible. You will also allocate space for local variables on the stack. This will give the experience of creating and using Stack Frames (also known as Activation Record). Please note that output parameters are almost always passed using registers.

2) Additionally, in previous labs you learned how to use unsigned variables (WORD, DWORD, etc) and unsigned arithmetic instructions (MUL and DIV) for unsigned multiplication and division. In this lab, you are going to demonstrate how to use *signed variables* (SWORD, SDWORD, etc) and signed arithmetic instructions (*IMUL* and *IDIV*) for unsigned multiplication and division. By defining a variable as signed, you can assign positive or negative numbers to it.

**Major Modifications of Lab 7 - HINTS**

CalculateW Procedure: This procedure requires the most modification. It should handle parameters as well as local variables all from the stack. It should also use imul and idiv instructions. I am going to give a hint to help some of you deal with the modifications seamlessly. If the CalculateW procedure you created in Lab 7 used registers to replace local variables for Loc1, Loc2, Loc3, and Sum, I recommend modification as follows. Replace the CalculateW procedure in Lab 7 with CalculateW procedure you used in Lab6 (with the MOD value changed from 7 to 3907). On the other hand, if the CalculateW you used in lab7 created the local variables within the CalculateW procedure, you must remove all the declaration statements because the stack will be used to create and access these local variables. In all cases Loc1, Loc2, Loc3, and Sum should not be seen anywhere in your program. Please check the requirements section below for further modification hints.

OutW Procedure: This procedure is modified from Lab7 with ability to use stack parameters as well as handle and print negative numbers. Check requirements section below for detailed algorithm

GetInput Procedure: This procedure’s parameters are passed via registers, so there should be minor changes if any. In general, there shouldn’t be any changes from lab7

**Requirements:** You should modify Lab 7 as follows.

1. **Multiple Decimal Digit Output (Positive or Negative Numbers)**: The input parameter for this procedure will be on the Stack so you should access it using the appropriate Stack Address. Recall 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 efficiently. In order to print both negative and positive numbers, modify the outW procedure you used in lab7 as follows.

1. Check to see if the number is positive

2. If yes, Go To Step 5

3. Otherwise, it is negative so convert it to positive Number (use NEG Instruction)

4. Print ‘-‘ character using WriteChar

5. Convert to Decimal Digits and Print as Usual as shown in the 10 steps below. This part should remain the same as in Lab7

1. Set a counter to zero. This counter will keep track of how many remainder decimal digits placed on the stack
2. Get number into the dividend registers for proper size.
3. Zero out remainder register.
4. Divide the dividend by 10 (use 32 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 remainder digits saved on stack
8. Test the quotient (next dividend) to see if it is zero.
9. If No then go to step 3
10. If Yes print the digits from the stack one by one. (Use POP instruction and Irvine library call WriteChar to print each number to the screen.)
11. **Variable Number of Input Decimal Digits terminated with Enter Key**: Keep this procedure (getInput) the same as in Lab 7. That is, *X* and *Y* are going to remain unsigned
12. **CalculateW : There are 2 major modifications you should make in this procedure.** 
    1. This procedure will use imul and idiv instructions to do signed arithmetic. To do so, replace mul with imul. In the case of signed division (idiv) we have to extend the sign of the dividend from eax through edx register instead of zeroing out edx by using mov edx,0 instruction. We extend the sign through edx by using the x86 instruction cdq (Convert Double to Quad) before using idiv. Therefore, replace all instances of mov edx,0 with cdq (Make sure cdq instruction comes right after you put the dividend in eax. Then replace all instances of div with idiv.
    2. Replace all instances of parameters and local variables with appropriate Stack frame addresses
13. **Main Program:** I will provide the main program and skeleton of the procedures. Consequently, as mentioned above you are creating 3 procedures. GetInput will be the same as in Lab7. CalculateW and OutW will be modified as specified above. The main Program stays in a loop until ‘q’ (quit) key is pressed.

**Procedure:** The main program does the following similar to Lab7: 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 the 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 as follows:

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. This could potentially be negative
4. Output W. If W is negative it should be printed as such.
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;

Do { *…*

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

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

W = CalculateW(X,Y) ;compute W

outW(W); ; 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

loc3 = loc2 – loc1

n=3000

n=n-1

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

*Return (* ***sum % 3907 + 3****);sum = sum* ***MOD*** *3907.* ***Note formula change!!***

*}*

outW(int W)

{

…

*Lines of code to output W (potentially multidigit decimal which can be positive or negative) to the screen.*See“Multiple Decimal Digit Output” *under requirements above for the Algorithm*

…

}

**A typical session should 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 A and B (if you used it) which are actually constants. This includes prompt strings defined the main program in the .data area. As you will notice, these string variables are all accessed in the main program. Note that the main program contains Calls to CrLf and WriteString so these Irvine procedures should not be in any of the procedures you are creating except WriteString in getInput for printing the prompts.

**Note2**: You may not use any library functions in your procedures other than Irvine Library functions *ReadChar* and *WriteChar and WriteString (used only in getInput procedure).*

**Note3**: Your program should be able to handle **variable number of unsigned decimal digits** for X, Y, and W just as in lab 7. X and Y are expected to be positive numbers, but **W can be negative**, depending on values of X and Y typed on the keyboard. If W is negative, it should be printed on the screen with a negative sign in front of it.

**Testing:** Please test your program and be sure it works before you submit it to CANVAS as specified below.

**Demonstration**: Demonstrate your program to the instructor with several screen shots similar to the session above. GRADER WILL CHECK your source code for 1) Whether parameters and local variables are used properly from the Stack; 2) Whether global variables are avoided in procedures except for A and B as specified above; 3) Whether your program handles multiple digit outputs in a procedure capable of printing both positive and negative numbers. That is, with negative number printed using a negative sign.; 4) Whether procedures written are well-documented; 5) whether your procedures work with the supplied main program under different conditions. For instance, if the grader chooses any positive values for X and Y, your getInput procedure should convert them to the correct binary values, CalculateW procedure should calculate W properly using parameters retrieved from the stack and local variables allocated and accessed from the Stack. Additionally, calculations must involve imul and idiv instructions. Finally, OutW procedure must print the output decimal number properly, with negative sign in front of it if the result is negative. 6) whether you used any Irivine Library procedures illegally. 7) The grader will assemble and run the documented source code you upload to CANVAS as specified below. The grader will check whether your program adheres to the session above as specified by your screen shots.

**Demonstration**: Demonstrate your program to the instructor by supplying screen shots for your session. It must be a Word Document. The Instructor will check whether all the procedures have been written, they are well-documented, whether they work with the supplied main program under different conditions, and whether you used any Irvine Library procedures illegally.

**Submission:** Submit well documented electronic copy of your program to CANVAS including the source code (.asm ), Output session, and report (.doc, docx). The source code should include the main program and the *3* procedures. The report should include program design for your procedures (in pseudocode and/or flowchart), complete stack frames for both CalculateW and outW, lessons learned, problems encountered, how long it took you to complete this assignment, and suggestions (if any) for the future. The output should include sessions you used to test it. As usual, **Filenames must be according to the format specified in the syllabus.**

**General Grading Rubric**

Demonstration: 50%

Report: 50% total will be based on

* Source Code -documented with comments: 35%. You will *lose 5% for each global variable* seen in any procedure.
* Stack frames: 5%
* Output Sessions: 5%
* Pseudocode/Flowchart and Comments (ie how long, problems, lessons learned, suggestions, etc): 5%