SP 2021 CSE 2421 LAB 7

Assigned: Friday, April 2nd

Early Due Date: Friday, April 16th, by 11:30 p.m.

Due: Wednesday, April 21st, by 11:30 p.m.

CAUTION: The Carmen assignment for this lab will only allow you to submit your lab ONE TIME! You must ensure that what you submit is exactly what you want to submit. No resubmissions will be accepted. This will allow the graders to grade your lab as soon as possible rather than waiting until the due date!

Objectives:

- Assembly language programs in x86-64
- Using assembler directives in x86-64
- Writing code for functions in x86-64; passing parameters in accordance with System V ABI conventions
- Working with values smaller than 8 bytes
- integrating C and assembler functions
- working with structures and arrays of structures in assembly language

REMINDERS and GRADING CRITERIA:

- Every lab requires a Readme file (for this lab, it must be called lab7Readme – the name is important, and case matters! This file should include the following:
    1. Certification:
       BY SUBMITTING THIS FILE TO CARMEN, I CERTIFY THAT I STRICTLY ADHERED TO THE TENURES OF THE OHIO STATE UNIVERSITY'S ACADEMIC INTEGRITY POLICY.
       THIS IS THE README FILE FOR LAB 7.
    2. Student name
    3. Total amount of time to complete the entire lab
    4. How you used gdb to help with your program specify breakpoints that you set, registers that you looked at and values that you found that allowed you to find your bug.
    5. One thing you learned or observed about programming in x86-64 while doing the lab.
    6. The two items specified below that belong in the README file
    7. Describe - to the best of your ability- the difference (if there is one) between the two 3D graphs that gnuplot creates from your data with 250 as the high value for x and y. If the graphs are consistent, say that.

- You should aim to always hand an assignment in on time. . If you are late (even by less than a minute), you will receive 75% of your earned points for the designated grade as long as the assignment is submitted by 11:30 pm the following day, based on the due date given above. If you are more than 24 hours late, you will receive a zero for the assignment and your assignment will not be graded at all. I encourage you strongly to start early – try to make the early submission date. Get your extra points! The most common reason for failing to submit on time is waiting too long to start!

- Any lab submitted that does not assemble (using your Makefile) or run without seg faults WILL RECEIVE AN AUTOMATIC GRADE OF ZERO for the lab. NO EXCEPTIONS will be made for this rule - to achieve even a single point on a lab, your code must minimally build (assemble to an executable) on stdlinux and execute on stdlinux without crashing, freezing, or causing segmentation faults. Your Makefile will be used to compile your code. You are responsible for making sure that your lab submits and compiles correctly.

  Since a Makefile is required for this lab, you must create the appropriate compile statements to create your 2 executable programs. Graders will be downloading your lab7.zip file from Carmen, unzipping it, and then executing **make** from a linux command line prompt. Your program must compile –without errors or warnings – via commands within the Makefile. **Given valid input as described below, your program must also run without having a seg fault or other abnormal termination.**

You are required to include comments in the code documenting what the code does and pay attention to formatting to make sure the code is as clear as possible and easy for a reader of your program to understand. The quality and clarity of the comments will be given weight in the grade for the lab.

LAB DESCRIPTION

You must write 2 versions of a program, **calc_lvalues (and calc_intvalues)**, which require 3 command line parameters: high value of x, high value of y, filename for results. The programs are to be written partially in C and partially in x86-64 assembler and calculate the following equation:

$$z = 13x^2 + 28x^2y^2 + 9y^2$$

for all values of x and y where –arg1 <= x <= arg1 and –arg2 <= y <= arg2

All (x,y,z) coordinate triples should be written to a file (arg3), in the format x y z, where x and y and y and z are separated by 1 space (One space only, Vasily!) and z is followed by a newline. **THIS FORMAT IS SPECIFIC AND REQUIRED!**

calc_lvalues will be **using an 8-byte unsigned value for z and 4-byte signed values for x and y** while **calc_intvalues** will be **using a 4-byte unsigned value for z and 4-byte signed values for x and y**. **Don't start on** calc_intvalues **until you have** calc_lvalues **working!**

```
struct ThreeD_values{
        unsigned long z;
        int y;
        int x;
};
```

Use this structure declaration exactly as shown.

You should use the partial_calc_lvalues.c and SP21.mult.s files currently posted to Piazza as a starting point for your **calc_lvalues()** program. The supplied program looks for 2 command line parameters (high value of x and high value of y) then passes them to mult(). The program, mult(), in SP21.mult.s, gives you the framework for a double nested loop, where -arg1<=x<=arg1 and -arg2<=y<=arg2, and (hopefully) enough comments so that you can understand what code goes where.
if mult() were coded in C, it might look like:

```
mult(int x, int y){
        long m, n;
        unsigned long z;
        for(m=-x; m <= x; m++){
                for(n=-y; n<=y; n++){
                        /* calculate z = 13x^2 + 28x^2y^2 + 9y^2 and then
                        Store x
                        Store y
                        Store z in appropriate structure of the array */
                }
        }
return;
}
```


Change the name of the partial_calc_lvalues.c program to calc_lvalues.c (and later calc_intvalues.c), change the name of SP21.mult.s to multlong.s (and later multint.s). **Any submissions with incorrect filenames will receive a 35-point deduction.**

malloc() a block of memory large enough to hold all the (x,y,z) coordinates you are about to create. **Determining the correct size of this block of memory is part of the lab.** Obviously, it changes based on the max values of x and y from the command line. The variable **limit** is declared to hold this value.

Once you have malloc()'d enough space, you will then call a function, multlong(), with the following prototype declaration:

void multlong(int a, int b, struct ThreeD_values * values);

where a=high x value, b=high y value, values = address you got from malloc().

You can use **one** multiply instruction to calculate x^2, **one** multiply instruction to calculate y^2, and **one** multiply instruction to calculate x^2y^2. **Make sure you use the correct multiply instructions for the int values that results in an 8-byte results.** All other calculations must be done with leaq, shifts, adds, etc**. If there are more than 3 multiply instructions in your code, you will receive a 50-point reduction. The graders will execute a grep mul *.s instruction on your code. If more than 3 multiply instructions show up in either multlong.s or mult_int.s, -50. Make sure your logic doesn't require more than one instruction.**

multlong() must populate each ThreeD_values structure element with the current values of x, y, and z while looping through all z calculations.

Upon return to main() (located in calc_lvalues.c), main() will open the filename of the 3rd command line parameter and write the x,y,z values on consecutive lines delimited by one space between values. **Make sure that you have the correct format string to print out signed and unsigned long values.** An example output file for the command

calc_lvalues 4 4 Results

will posted on Piazza by 4/6; most likely it will be much earlier than that.

For the test case, you should run calc_lvalues with values for x and y of 50, 100, 200, and 250 each. Give each of the results a different file name so you can compare the plots.

From a Linux prompt bring up the program **gnuplot**.

At the gnuplot> prompt, type:

splot "FileName", where **FileName** is one of the files you created with calc_lvalues. The double quotes around the filename are required. Then, do the same with each of the other filesnames you created. Note in your README file whether the plots are consistent or change as the number of points increases.

Once you have calc_lvalues working to your satisfaction, then copy calc_lvalues.c to calc_intvalues.c and multlong.s to multint.s. Edit calc_intvalues.c and change the ThreeD_values structure to:


```
struct ThreeD_values{
        unsigned int z;
        int y;
        int x;
};
```


Use this structure declaration exactly as shown.

Other changes you will have to make in calc_intvalues.c are: 1) size of the area to malloc() changes, 2) must change multlong() call to multint() call, 3) must change format of the printf string to print out unsigned integers instead of signed/unsigned ints/longs.

Changes you will have to make in multint() are: 1) many instruction suffixes will change because you are now working with 4 byte values, 2) Size of all registers with x, y, or z values in them must be converted to 4-byte registers, 3) the multiplication instructions must change, 4) address calculations when writing to dynamic memory have changed.

You will have to modify your Makefile to create a calc_intvalues executable as well as calc_lvalues executable. HINT: 1st line of Makefile changes to

all: lab7.zip calc_lvalues calc_intvalues

I THINK that these are all the changes that must be made, you may find others that I have forgotten since I implemented this.

For the test case, you should run calc_intvalues with values for x and y of 50, 100, 200, and 250 each. Give each of the results a different file name so you can compare the plots.

From a Linux prompt bring up the program **gnuplot**.

At the gnuplot> prompt, type:

splot "FileName", where **FileName** is one of the files you created with calc_intvalues. The double quotes around the filename are required. Then, do the same with each of the other filesnames you created. Note in your README file whether the plots are consistent or change as the number of points increases.

Now you can answer the last question in the lab7Readme file. Are the graphs you plotted using unsigned long z values consistent with the graphs plotted using unsigned integer z values? If they are consistent, why do you think that is? If they are not consistent, why do you think that is?

CONSTRAINTS:

To pass parameters to functions, you must follow System V ABI conventions.

You must also follow the System V ABI conventions related to caller and callee saved registers, caller cleanup, and returning values from procedures in register rax.

REQUIREMENTS

1. Required file names: **calc_lvalues, calc_intvalues, calc_lvalues.c, calc_intvalues.c, multlong(), and multint()**.
2. Within your x86-64 programs:
   a. You may only use x86-64 constructs that we have discussed in class, you can find in the slides, or you can find in the sample x86-64 programs posted on Piazza.
   b. You may not store any values to the stack other than by using the push/pop instructions.
   c. Only values that can be put on the stack are for caller/callee saved register purposes.
   d. You must have all working values in registers – none stored on the stack.
   e. You must use correct stack frame procedures.
   f. You must use all (needful) x86-64 directives.
   g. You must use the correct suffix for all data types.
   h. You must use correct memory addressing modes.
   i. You must use correct caller/callee saved register conventions.
   j. You must store the value for x, then the value of y and last the value of z in each structure – **in that order** within each iteration of the loop.
3. You must comment your code!
4. Your Makefile, multlong.s and multint.s files submitted to Carmen as a part of this program **must** include the following at the top:

```
#BY SUBMITTING THIS FILE TO CARMEN, I CERTIFY THAT I STRICTLY ADHERED TO THE
#TENURES OF THE OHIO STATE UNIVERSITY'S ACADEMIC INTEGRITY POLICY.
# Name: <your name goes here>
```

5. Your calc_lvalues.c and calc_intvalues.c files submitted to Carmen as a part of this program must include the following at the top:

```
/* BY SUBMITTING THIS FILE TO CARMEN, I CERTIFY THAT I STRICTLY ADHERED TO THE
TENURES OF THE OHIO STATE UNIVERSITY'S ACADEMIC INTEGRITY POLICY.
Name: <your name goes here>
*/
```

6. Your calc_lvalues.c and calc_intvalues.c files submitted to Carmen as a part of this program must have appropriate error checking to validate the correct number of command line parameters and an associated USAGE: error message if they are not present.

7. Both **calc_lvalues** and **calc_intvalues** will be evaluated with **valgrind** to check for memory leaks.

LAB SUBMISSION

You have submitted enough labs by this time that understanding how to do it should not need a detailed explanation. I highly recommend once you have created your .zip file to submit, that you create a test directory within your lab7 directory. Then copy your .zip file there, cd to the test directory, unzip your file and run make. If your make command generates any errors/warnings or doesn't create the two needed executable files, then the .zip file you were planning to submit to Carmen is incorrect.