SP 2022 CSE 2421 LAB 7

Assigned: Thursday, November 10th
Early, Early Due Date (25% Bonus): Friday, December 2nd , by 11:30 p.m.
Early Due Date (10% Bonus): Monday, December 5th by noon
Due: Tuesday, December 6th, by 11:30 p.m.

IMPORTANT: READ THESE INSTRUCTIONS AND FOLLOW THEM CAREFULLY.

Objectives:

- Command line argument processing using x86-64
- Using an array of structures in x86-64
- Calling C library functions within x86-64 program
- correctly using different size registers
- looping within an x86-64 program
- working with add, subtract, multiply and divide in x86-64

REMINDERS and GRADING CRITERIA:

➢ This is an individual lab. No partners are permitted.

➢ Start now! There will be no deadline extensions for this lab. Plan accordingly.

➢ **Every lab requires a Readme file** (for this lab, it should be called **lab7Readme – use only this name.**). This file must be a simple text file created on stdlinux that includes the following:

· Disclaimer:
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.

· Your name

· Total amount of time (effort) it took for you to complete the lab

· Short description of any concerns, interesting problems or discoveries encountered, or comments in general about the contents of the lab

· Describe how you used gdb to find a bug in your x86-64 programs. Include details with respect to what breakpoints you set what registers you looked at and how you determined what the bug was.

➢ You should aim to always hand an assignment in on time or early. If you are late, you will receive 75% of your earned points for the designated grade if 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.

➢ Any lab submitted that does not compile/make – ***without errors or warnings*** - and run without seg faults or exhibit any other major OS errors/faults **WILL RECEIVE AN AUTOMATIC GRADE OF ZERO**. No exceptions will be made for this rule - to achieve even a single point on a lab, your code must minimally build (compile to an executable) on stdlinux and execute on stdlinux without hanging or crashing, using the following command: **make**. A seg fault is considered crashing.

LAB DESCRIPTION

All functions must be written in x86-64 assembly language.

1. You will be writing a program called **readrec** that will expect 2 command line parameters. The first argument is the number of records to read, and the second argument is the filename from which to read the records. Within the main() function:

   a. Dynamically allocate space for X number of these structures:

   ```
   struct Record{
                   int value2;
                   int value1;
                   int difference;
                   long sum;
                   long product;
                   int quotient;
                   int remainder;
   };
   ```

   where X is the value of the first command line argument.

   b. Using fopen(), open the filename specified by the second command line argument. There will be ***at least*** X number of lines in the file. Each line will hold 2 values separated by whitespace.

   c. Write and then call an x86-64 function called readlines(). If it were written in C, the prototype would look like this:

   void readlines(FILE *fptr, struct Record *rptr, int count);

   This function will use fscanf() to read **count** lines from **fptr**, one by one, and store the first value in structure member value1, the second value in structure member value2.

   d. After reading in all values close the file using fclose().

   e. Write and then call an x86-64 function called compute(). If it were written in C, the prototype would look like this:

   void compute(struct Record *rptr, int count);

   Using value1 and value2, this function will compute the sum (value1+value2), the difference (value1-value2), the product (value1*value2), and quotient/remainder(value1/value2). Both value1 and value2 can be positive or negative, while value1 can also be zero. Note that sum and product are twice the size of value1 and value2.

   f. Write and then call, an x86-64 function called printlines(). If it were written in C, the prototype would look like this:

   void printlines(struct Record *rptr, int count);

   This function will use printf() to print out all values within the structure array in this format:

   value1 + value2 = sum

value1 – value2 = difference

value1 * value2 = product

value1/value2 = quotient, remainder

g. Free the dynamically allocated memory.

2. You must create a Makefile where **all** will have 2 targets: readrec and lab7.zip.

REQUIREMENTS

1. You must have a separate .s file for each function you write.
2. 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. Any other x86-64 constructs will invalidate your lab submission.
3. You must use correct stack frame procedures and conventions.
4. No values in the x86-64 functions of your program may store values on the stack other than with push/pop instructions.
5. No values in the x86-64 functions of your program may store values on the heap. (Memory dynamically allocated via the malloc/calloc does not apply with respect to this requirement.)
6. You must use all (needful) x86-64 directives.
7. You must use the correct suffix for all data types (instructions and register sizes).
8. You must use correct caller/callee saved register conventions.
9. You must include the certification header in your .s files.
10. You must comment your code!

HELPFUL SUGGESTIONS

1. Remember that you have **int argc** and char ****argv** in registers %rdi and %rsi, respectively, when you enter main().
2. Don't forget to use **tui reg general** when in gdb to see register values.
3. You can use **next** in gdb when you want **do not** want to go in to the code of the function being called (printf, for example).
4. You can use **step** in gdb when you want **do** want to go into the code of the function being called.
5. You can reference http://csapp.cs.cmu.edu/public/docs/gdbnotes-x86-64.pdf for several other very helpful gdb instructions for x86-64 use.

LAB SUBMISSION

You must submit all your lab assignments electronically to Carmen in .zip file format. You must create your zip file from within your Makefile.

I highly recommend testing your .zip submission you've created prior to submission. If things are missing, your lab can not be graded, and you will receive a 0 on the lab.

NOTE:

- Your programs MUST be submitted in source code form. Make sure that you zip all the required files for the current lab (and .h files when necessary), and any other files specified in the assignment description. Do NOT submit the object files (.o) and/or the executable. The grader will not use executables that you submit anyway.
- It is YOUR responsibility to make sure your code can compile and run on CSE department server **stdlinux.cse.ohio-state.edu,** without generating any errors or warnings or segmentation faults, etc. Any program that generates errors or warnings when compile or does not run without system errors will receive 0 points. No exceptions!