CSE 2421 – Systems 1

Spring Semester 2024

Programming Assignment #6

- **This assignment is worth 10pts.**
- **You must upload the zipped solution folder to Carmen, as solution.zip.**
- **The deadline for this assignment is April 19th 11:59pm ET.**
- **No late submissions will be accepted.**
- **The main topics of this assignment are: use ofaddress modes and function calls in x86-64 assembly.**

Preliminary Instructions:

- Download the entire folder a6 from Carmen.
- Recall that assembly programming is system dependent (architecture, processor, OS, etc). To avoid portability and setup problems, all of your work should be done exclusively in stdlinux or coelinux. You should not work in your own laptop or desktop, unless it is a Linux OS with an Intel processor.
- After downloading the folder a6, copy the entire folder to your favorite OSU linux cluster (stdlinux or coelinux).

Instructions:

The goal of this assignment is to reproduce, in x86-64 assembly, the functionality of the program sort-vector.c. Compile, build and run this program to see its output. Read the C file provided to understand the functionality that you should replicate in assembly.

You are provided with a template assembly file, `sort-vector.asm`. This file defines a global array of integers, 9 in total, and several global variables (including induction variables and variables with messages to print).

The template file provides already three auxiliary functions: `print_digit`, `print_array` and `print_message`.

You are asked to complete the implementation of the program by writing two more auxiliary functions: `must_swap` and `sort_from_ii`. See file sort-vector.c to understand their implementation.

- [2 pt] `must_swap(x,y)` takes the values of two arguments and returns 1 if the second argument is less than the first one. Otherwise, the function returns 0. You must use this function to determine when two values must be switched. Test this function with any pair of values before proceeding to the next part. You can use any mechanism to pass parameters to your function. However, it is recommended to use only registers and to follow the conventions for function calls that we have discussed through the course. Return the result through the `rax` register.

- [3 pt] `sort_from_ii(ii)`: See the c implementation of this function. This function must make use of must_swap. sort_from_ii does not return any value.

[3 pt] In the main function of the program, you must iterate from 0 to 7 (including), and call, for each iteration, the assembly function `sort_from_ii`.

Recall that some registers have dedicated (hardwired) roles or purpose on the caller and callee functions. Similarly, by convention, some registers must be used when passing parameters and invoking system calls. The following table summarizes these considerations.

%rip	Instruction pointer
%rsp	Stack pointer
%rax	Return value
%rdi	1st argument
%rsi	2nd argument
%rdx	3rd argument
%rcx	4th argument
%r8	5th argument
%r9	6th argument
%r10,%r11	Callee-owned
%rbx,%rbp, %r12-%15	Caller-owned

[2 pt] Comment your code adequately. You should describe the role of each register, the type of address mode used, addresses you attempt to calculate, and any calls to your functions.

Suggestions and Reminders:

- Do not use caller-owned registers in the callee function, and do not use callee-owned registers in the caller function (unless you are setting a value in the register).

- To simplify the program logic, do not assume that your registers hold concrete values before and after functions calls. In particular, when handling induction variables, always read the most recent value from memory, and upon updating it, write the new value to memory. For your convenience, a few induction variables are already declared in the .data segment.

- In all implemented functions, remember to store the current value of the register base pointer in the stack, copy the register stack pointer to the base pointer, and to perform `leave` and `ret` at the end of each function.

What and where to upload:

- Upload via Assignments->Assignment 6 in Carmen.
- You must upload a single compressed folder named solutions.zip. The folder should contain the assembly file sort-vector.asm, and a screenshot showing the output you obtain.
- The screenshot should show you logged into a linux cluster, your username, the date and the output of your assembly program (via the build.sh script or by doing echo $?).
- Your programs should run in either stdlinux or coelinux.