CS244, STELLENBOSCH UNIVERSITY

TUT 4: RECURSION AND CALL-BY-REFERENCE PARAMETERS IN 32-BIT INTEL ASSEMBLY LANGUAGE

15 OCTOBER 2021

> Software gets slower faster than hardware gets faster. (Sometimes formulated as Grove [of Intel] giveth and Gates [of Microsoft] taketh away.)
>
> —Niklaus Wirth

OVERVIEW

The goal of this tutorial is to get acquainted with writing recursive functions in assembly language. Some of these functions may use call-by-reference parameters. After successfully completing the tutorial, you should be able to:

- translate recursive C functions to assembly language by hand;
- understand how the C function call convention `cdecl`, function results, and call-by-value and call-by-reference parameters work in both C and assembly language; and
- use the Intel assembly language instructions `add`, `call`, `cmp`, `dec`, `imul`, `inc`, `j`*cc*, `jmp`, `lea`, `mov`, `pop`, `push`, `ret`, `shr`, and `xchg`.

A NOTE ON 32-BIT V. 64-BIT EXECUTABLES

Remember that we create 32-bit executables. If, for example, you use the `sizeof` operator in a C program to discover the number of bytes a data structure or variable occupies, then you must compile that program with the `-m32` switch. The makefile included in the repository already has this switch included in the compilation directive.

INSTRUCTIONS

1. Clone your repository for Tutorial 4 at

   `rw244-2021@gitolite.cs.sun.ac.za:`⟨*US number*⟩`/tut4`

2. The repository contains a number of files, including a makefile and a C program that must be used for testing. The C header file `tut4.h` contains the following five prototypes:

   - `int fact(int n);`
   - `int ackerman(int x, int y);`
   - `int power(int x, int y);`
   - `void swap(int *x, int *y);`
   - `int binary_search(int n, int list[], int low, int high);`

The `swap` function is already implemented and should be used as a guideline for completing the `binary_search` function since both use call-by-reference parameters in the form of pointers/arrays. Complete the remaining four functions in assembly language using the C implementations in the file `test4.c`. *You do not need local variables in your assembly language functions.*

3. Because of the computational cost of recursive calls, the C implementations for `fact`, `power`, and `binary_search` functions are not particularly efficient. In practice, tail recursion in imperative languages – like C, Python, and Java – should be replaced by simple looping. However, because this tutorial is about recursion, *you have to stick to the recursive approach in your assembly language implementations.*

4. You have to complete `binary_search` for submission via Git. The last commit will be assessed as your attendance assignment for next week.

5. You MUST code your solutions by hand, and you MAY NOT use any tools for disassembly or decompilation. You MUST complete your plagiarism declaration in the repository.

6. There are no marks for style, but marks may be subtracted for using local variables and for particularly baroque attempts.

7. For the sake of interest: The Ackermann function is the simplest example of a well-defined total function which is computable but not primitive recursive. Refer to the following link for more information: `http://mathworld.wolfram.com/AckermannFunction.html`. You should also find the following blog post useful for your binary sort implementation: `https://ai.googleblog.com/2007/06/extra-extra-read-all-about-it-nearly.html`.