P4A - Assembly Language Programming

New Attempt

Due Tuesday by 11:59pm **Points** 30 **Submitting** a file upload **File Types** s
Available Jul 26 at 12am - Aug 9 at 11:59pm 15 days

Project 4A: Assembly Language Programming

This project will help you gain familiarity with writing X86 Assembly Language Code.

Tips to Get Started

I'm adding this section after meeting with several students during office hours.

This project has an extremely difficult learning curve if it's your first experience writing assembly code. However, once you figure out the basics of assembly programming it's not difficult and doesn't take long to finish, but the beginning can be extremely frustrating. You're going to invest hours learning but then only a few minutes writing the code. Start by watching all of the videos. Everyone I talked to during office hours who was struggling had not watched all of the videos about assembly. Start with the videos, then read the book. After you're caught up with the learning part of the project - then write the code.

Corrections and Additions

1. Please do not rename the files. Please submit files that are exactly named "count.S and sort.S".

2. CSL Machines represent negative numbers in two's complement with exactly 32 bits. Please count all ones regardless of whether the number is positive or negative. Testing with -1 should return 32 ones.

Video: Debugging this project with GDB

Specifications

For this project, we will be writing two small programs:

1) **Count the '1's in a binary representation of a number.** You need two files for this program. ones.c is the C code driver that will call the function you write and print out how many of the binary digits in a number are 1s. For example, the 4-bit binary representation of 6 is 0110 which has two '1's. Write your code in assembly in count.S. This file represents the assembly language code defining the function call to countOnes(). See the function prototype in ones.c. We recommend using a loop, bit masking with the bitwise & operator, and bit shifting.

2) **Sort a list of numbers.** For this program, you will be finishing the main function found in sort.S. Write your code in assembly. The provided template file calls getArr to generate an array of 20 pseudorandom integers and then calls printArr to print them out. Your task is to complete the program by sorting the list of numbers before they are printed. You are welcome to use any sorting algorithm you like, but we recommend **Bubble Sort (https://en.wikipedia.org/wiki/Bubble_sort)** for your first assembly programming experience. Here is a **C version**

(https://www.geeksforgeeks.org/bubble-sort/) of the code. Grading here is results-based; there is no need to include any optimizations to the sorting algorithm. Sorting a list with only 20 numbers goes really fast on a modern computer even with a low-efficiency algorithm. You may write your code directly in main - there is no need to write extra functions.

- **For each program write a C version of your program as a comment at the top of the file.**
- **Assembly code can be difficult to read. For every line add a comment explaining what it's doing.**
- **Write your own code! Do not write the code in C and then compile it to assembly. Do not use any call frame information (.cfi_) assembler directives.**
- **You are expected to complete this project on the CSL machines.**

Files

- **ones.c (https://canvas.wisc.edu/courses/248855/files/20816996/download?download_frd=1) count.S (https://canvas.wisc.edu/courses/248855/files/20816998/download?download_frd=1)**
- **nums.c (https://canvas.wisc.edu/courses/248855/files/20816995/download?download_frd=1) sort.S (https://canvas.wisc.edu/courses/248855/files/20816997/download?download_frd=1)**

Compiling and Running on the CSL Machines

```
(the default output file name is a.out if the -o option is left unspecified)
>>> gcc count.S ones.c -m32 -Wall
>>> ./a.out

>>> gcc sort.S nums.c -m32 -Wall
>>> ./a.out
```

Strategy

Write your code in small pieces. **One line at a time.**

Both programs already print out numbers. Use that to your advantage to print out intermediate results.

Avoid the temptation to search for assembly algorithms on the internet. You will learn best by struggling with this project on your own. It's too easy to write someone else's solution after just looking at it for short programs like these.

Also, avoid the temptation to write a C version and then compile it to assembly and look at the C compiler's solution to the program. We expect you to do your own work. It is very easy for us to see the difference between compiler-generated assembly code and student-written code.

In the past, students have reported feeling extremely frustrated as they started this type of project, then once they got through enough of the learning curve that it clicked it only took a few minutes to finish. Plan to spend several hours "learning" and only a short time "writing".

Shift operations can only use constants like $1 or the %cl register.

Recall that %edi, %esi, and %ebx are callee saved registers. I recommend using these registers, but be sure to restore their contents before the function finishes or your code will crash later.

Resources

http://pages.cs.wisc.edu/~powerjg/cs354-fall15/Handouts/Handout-x86-cheat-sheet.pdf
http://www.cs.cmu.edu/afs/cs/academic/class/15213-s20/www/recitations/x86-cheat-sheet.pdf

Turn in

Upload both your count.S and sort.S files. Double-check to make sure you turned in the correct file. Do not rename the files.

Style

0. **Comment!!!**

1. Use **indentation**

2. Align the assembly commands, operands, and comments in **three columns**.

3. Break your code into **blocks** separated by blank lines.