California State University, Sacramento
College of Engineering and Computer Science

Computer Science 35: Introduction to Computer Architecture

Fall 2020 – Lab 2 – *Election Time*

Overview

It's Election Time. I know, I know. You are tired the endless negativity, the endless media bias, the endless… you get the idea.

Anyway, let's have fun in a stressful time. During the elections, candidates fill a wide variety of offices – from President to Major to Supervisor. You are going to create a program that does one of the things computers were designed to do – math! In particular, let's calculate how many votes our candidate has earned.

Winning a campaign is difficult. You have to go out and gain supporters – the ones you know will vote, put out lawn signs, etc…

For this program, you are going to input four things that will either cause the candidate to gain or lose votes and then display the net gain (or lose).

Your Task

You must think of a solution on your own.

Basically, you are going to input 4 values and store them in memory (using direct storage). Then, you are going to calculate the result by using the subtract (sub) and add instruction.

Sample Output

Your program does not have to look exactly like this, but this is an example of a working solution. **Have fun! Change the prompts to whatever you want. Be creative!**

The user's input is printed in **blue**. The data outputted from your calculations is printed in **green**. You don't have to make the text that color in your program.

```
Votes gained from lawn signs.
550
Votes gained from mailers.
900
Votes lost from the other candidate's attack ads.
300
Votes lost from the debate.
100
The net change in votes is 1050
```

Tips

The Data Section

You need to create ASCII strings and quad integers in the data section. You should create a label for each value and prompt. The quad directive requires an initial value.

```
LawnSignPrompt:
   .ascii "Votes gained from lawn signs.\n\0"

LawnSign:
   .quad 0
```

Reading Integers

The CSC 35 Library has a subroutine called "ScanInt" that will read a value from the keyboard and store it into **`rbx`**. This is equivalent to the Java Scanner class method "nextInt".

```
call ScanInt            #rbx = scanner.nextInt();
```

Requirements

You must think of a solution on your own. The requirements are as follows:

1. Store each inputted value using direct storage.
   Remember: the names of the labels are up to you. **Any lab not using direct storage will receive a zero.**
2. Input all four values with a nice prompt for the user.
3. Compute the total – using additions and subtractions. You can get a negative result.
4. Output the result to the screen with some nice text

Submitting Your Lab

This activity may only be submitted in Intel Format.
Using AT&T format will result in a zero. Any work from a prior semester will receive a zero.

Afterwards, run Alpine by typing the following and, then, enter your username and password.

```
alpine
```

Please send an e-mail to yourself (on your Outlook, Google account) to check if Alpine is working. To submit your lab, send the assembly file (not a.out or the object file) to:

```
dcook@csus.edu
```

UNIX Commands

Editing

Action	Command	Notes
Edit File	`nano` *filename*	"Nano" is an easy to use text editor.
E-Mail	`alpine`	"Alpine" is text-based e-mail application. You will e-mail your assignments it.
Assemble File	`as -o` *object source*	Don't mix up the *object* and *source* fields. It will destroy your program!
Link File	`ld -o` *exe object(s)*	Link and create an executable file from one (or more) object files

Folder Navigation

Action	Command	Description
Change current folder	`cd` *foldername*	"Changes Directory"
Go to parent folder	`cd ..`	Think of it as the "back button".
Show current folder	`pwd`	Gives the current a file path
List files	`ls`	Lists the files in current directory.

File Organization

Action	Command	Description
Create folder	`mkdir` *foldername*	Folders are called directories in UNIX.
Copy file	`cp` *oldfile newfile*	Make a copy of an existing file
Move file	`mv` *filename foldername*	Moves a file to a destination folder
Rename file	`mv` *oldname newname*	Note: same command as "move".
Delete file	`rm` *filename*	Remove (delete) a file. There is **no** undo.