California State University, Sacramento
College of Engineering and Computer Science

Computer Science 35: Introduction to Computer Architecture

Fall 2021 – Lab 2 – *Gringotts Wizarding Bank*

Overview

Whether you are a student at the famous ***Hogwarts School of Witchcraft and Wizardry***, or a wizard working a humble job in Hogsmeade, you need to keep track of your finances.

The ***Gringotts Wizarding Bank*** is renowned throughout the Wizarding World is a the premier (and feared) banking firm.

Like the muggle world, wizards and witches need to keep track of how much money you have. Debt is bad.

Your Task

You are going to create a program that does one of the things computers were designed to do – math! Though you might not know yet, but most of your life will be devoted to balancing your books.

The goblins, like the muggles, use the simple equation to determine how much money you bring home each month:

Cash Flow = Income – Expenses

You are going to write a program that inputs 2 sources of income, 2 expenses, and then displays the user's cash flow. Basically, you are going to input 4 values and store them in memory (using direct storage). You must think of a solution on your own.

Sample Output

Your program does not have to look exactly like this, but this is an example of a working solution. The user's input is printed in **blue**. The data outputted from your calculations is printed in **red**. You don't have to make the text that color in your program.

```
Gringotts Wizarding Bank

How much to you earn from work?
1800

How much do you spend on food?
525

How much do you spend on rent?
350

How much do you spend on dark magic objects?
150

Your expenses are $1025

So, your cash flow is $775
```

Tips

Reading Integers

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

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

How to approach the problem

- Work in your program in parts
- The **rax** register is used by the library to input data and output results. Use direct storage store the values. In fact, you **must** use direct storage for credit.
- Pay close attention to which operand is changed with the SUB and ADD instructions.
- Intel x64 does **not** allow both operands to be address (labels).

Requirements

- You must think of a solution on your own.
- Any lab not using direct storage will receive a zero.
- Any lab containing conditional logic will receive a zero (we have not covered it yet).

The requirements are as follows:

1. Input one source of a helpful prompt.
2. Input **three** expenses with helpful prompts.
3. Compute and display the total expenses.
4. Compute and display the cash flow.

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.

To submit your lab, you must run Alpine by typing the following and, then, enter your username and password.

```
alpine
```

To submit your lab, send the assembly file (do not send the 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.