

CSE12 Lab 4: Simple CSV File Analysis

Due at 11:59 PM on the marked due date

Objective

The objective of this lab is to learn about function calling, RISC-V protocols for the use of registers.

This lab takes as input a CSV (comma separated values) file (.csv extension) that is used for generating tabular data in spreadsheets. **Specifically, for this assignment, you will NEED to assume this CSV file was generated under Windows** (the reason will be explained shortly). Consider the *data.csv* file below as it appears when you open it in Excel as an example.

	A	B
1	Kruger Industrial Smoothing	365
2	Kramerica	0
3	Vandelay Industries	500
4	Pendant Publishing	100
5	J. Peterman Catalog	42
6		

Assignment Project Exam Help

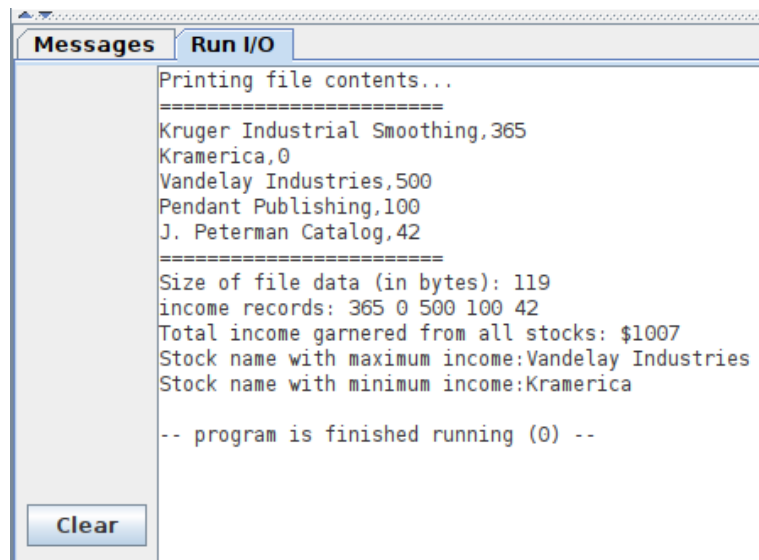
This file shows the stock returns from an investment portfolio over a year. The “A” column contains the stock name and the “B” column indicates the returns in USD (**We will assume that there is no negative stock return in all our CSV files**).

<https://powcoder.com>

You will run the file *lab4_testbench_rev##.asm* file in RARS which takes *data.csv* as an input CSV file. Doing so will yield the following analysis, based on the calculations made by the assembly files that you will be submitting):

1. Find the total file size in bytes (excluding any metadata generated by your OS). (*length_of_file.asm*)
2. List the dollar amount of all the input records. (*input_from_record.asm*)
3. Provide the name of the stock that gives the maximum income. (*maxIncome.asm*)
4. Provide the name of the stock that gives the minimum income. (*minIncome.asm*)
5. Calculate the total income generated from all stocks

When you run via RARS *lab4_testbench_rev#.asm* with the *.asm* files shown above completed by you, you will get the output console as shown below:



```
Messages Run I/O
Printing file contents...
=====
Kruger Industrial Smoothing,365
Kramerica,0
Vandelay Industries,500
Pendant Publishing,100
J. Peterman Catalog,42
=====
Size of file data (in bytes): 119
income records: 365 0 500 100 42
Total income garnered from all stocks: $1007
Stock name with maximum income:Vandelay Industries
Stock name with minimum income:Kramerica

-- program is finished running (0) --

Clear
```

Figure 2 After running the *lab4_testbench_rev##.asm* file with the Lab4 assignment fully completed

About the Windows CSV file format

To distinguish between each entry/row/record in the spreadsheet format of the CSV file, the following convention is adopted depending on the OS :

- Windows - Lines end with a <CR> and a <LF> character
- Linux - Lines end with only a <LF> character
- Macintosh (Mac OSX) - Lines end with only a <LF> character
- Macintosh (old) - Lines end with only a <CR> character

where <CR> is the carriage return ('`\r`') character and <LF> is the line feed/newline ('`\n`') character.

If you open the provided *data.csv* file in Notepad++ on Windows with “Show all Characters” enabled, then you should see the following text showing the placement of the carriage return and line feed characters. .

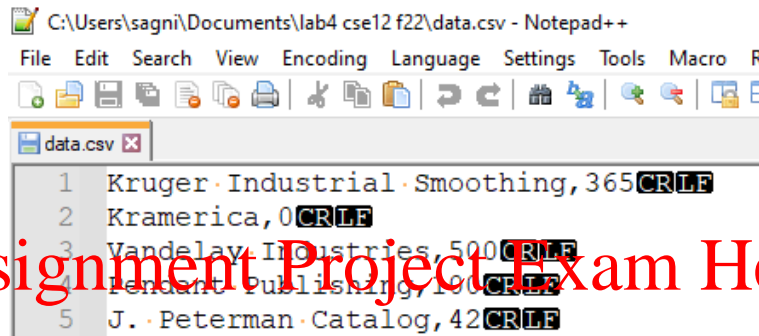


Figure 3 data.csv on Notepad++ on Windows

So, for example, if I were to express record 2 from *data.csv* as a string of characters in RARS, I would write: “Kramerica,0\r\n”. If you are using an OS that is NOT Windows, it is likely that *data.csv* would not open correctly due to the encoding differences. If you have a text editor like Notepad++ that allows you to see all characters, make sure that the “\r\n” appears for each record in the file as shown in Figure 3. This is the case for the *data.csv* that we include in the directory.

Another assumption that we will state at this point is that we expect that in each record, the name of the stock is followed by the “,” and then immediately by the stock price expressed as an unsigned integer in base 10. So, with record 2 as an example, we will never have a situation where it is written as “Kramerica, .. 0\r\n” where the 2 red dots indicate two blank spaces. This assumption makes the CSV file analysis by RARS easier to code.

Resources

Much like how a high-level program has a specific file extension (.c for C, .py for python) RARS based RISC-V programs have an .asm extension.

In the Lab4 folder in the course you will see 9 assembly files. They are meant to be read (*and understood*) **in sequence** and they will provide you with lotsof hints as to how to build your program:

1. *add_function.asm* – This program accepts two integers as user inputs and prints their addition result. The actual addition is done through calling a function, *sum*. *Sum* accepts two arguments in the a0, a1 registers and returns a0+a1 in a0 register
2. *multiply_function.asm* – This program accepts two integers as user inputs and prints their multiplication result. The actual multiplication is done through calling a function, *multiply*. *Multiply* accepts two arguments in the a0, a1 registers and returns a0*a1 in a0 register. This function in turn calls the function *sum*, described previously, to do a particular addition. Thus, *multiply* function is an example of a **nested function call**, a function which itself calls another function, *sum* in our case.
3. *The lecture slides provide a lot of information about register calling and saving conventions.* Studying the comments in *add_function.asm*, *multiply_function.as* alongside with the notes should be sufficient to create to allow

you to complete this assignment.

4. **lab4_testbench_rev##.asm** - This is the main testbench program you will run upon completion of all coding in Lab4 to ensure your Lab4 assignment works as expected. This file is initially provided such that if you run it as it is (with the other .asm files in the same directory), you will still get partially correctly generated output. This testbench will also run the the function *allocate_file_record_pointers* from the *allocate_file_record_pointers.asm* which will aide you in writing your program. **DO NOT MODIFY THIS FILE.**

5. **allocate_file_record_pointers.asm** - This .asm file contains a function that **creates an internal reference table** in memory to pointer pairs. These pointer pairs indicate 1) the location of the start of a string corresponding to the stock name and 2) the start of a location containing the stock price for each and every record/entry coming from the data.csv file. This function has been fully written out for you. **DO NOT MODIFY THIS FILE.**

6. **macros_rev#.asm** - *This file contains useful macros, mostly to do I/O, it uses and modifies a# registers, so be careful. Become familiar with these functions. They are your friends.*

6. **income_from_record.asm** - This. asm file contains a function that **you will write** to convert the string data from the income of a record/entry in the spreadsheet and convert it into an integer. Example, convert the string “1234” into the actual integer 1234 in base 10.

7. **income_from_record_ideas.asm** - This file provides some ideas for implementing income_from_record.asm.

7. **length_of_file.asm** - This .asm file contains a function that **you will write** to find the total amount of data bytes in the csv file. Refer to Figure 2 for an example.

8. **maxIncome.asm** - This. asm file contains a function that **you will write** to determine the name of the stock that has the maximum income in the csv file. Refer to Figure 2 for an example.

9. **minIncome.asm** - This. asm file contains a function that **you will write** to determine the name of the stock that has the minimum income in the csv file. (Figure 2 fails to show the corresponding output. It'll be added later)

10. **totalIncome.asm** - This. asm file contains a function that **you will write** to sum up all the stock incomes in the csv file. Refer to Figure 2 for an example.

Please download all these files and make sure to open them in the **RARS Text editor only**. Otherwise the comments and other important code sections may not be properly highlighted and it can be a hindrance to learning assembly language intuitively.

These files have enough comments in the source code to jump start your understanding of RISC-V assembly programming for Lab 4 if the lectures have not yet covered certain topics in assembly programming.

Beyond these three files, **you should have all the required resources in the Lecture Slides themselves. The slides are very self-explanatory and it is encouraged you start reading them even if the instructor hasn't started discussing them in lecture.**

For the usage of macros (which are utilized heavily in this lab to generate ecalls), please also refer to the RARS documentation on [macros](#) and [ecalls](#) as well.

Please read the provided files carefully. You can learn a lot about assembler from reading these files. Note that using macros resembles calling functions. The macros have been written to make use of the a# registers, so don't assume that any values in your a# registers remain valid after calling a macro.

Memory arrangement as defined in Lab4

The memory of RISC V is used as per the given requirements.

File Data Buffer

The **data.csv** file is treated as the default input file. It's contents are store in the file buffer location 0xffff0000 (referred to as MMIO).

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0xffff0000	g u r K	I r e	s u d n	a i r t	m S l	h t o o	, g n i	\r 5 6 3
0xffff0020	a r K \n	i r e m	0 , a c	a V \n \r	l e d n	I y a	s u d n	e i r t
0xffff0040	0 5 , s	P \n \r 0	a d n e	P t n	i l b u	n i h s	0 l , g	J \n \r 0
0xffff0060	e P .	m r e t	C n a	l a t a	4 , g o	\n \n \r 2	\n \n \n \n	\n \n \n \n
0xffff0080	\n \n \n \n	\n \n \n \n	\n \n \n \n	\n \n \n \n	\n \n \n \n	\n \n \n \n	\n \n \n \n	\n \n \n \n
0xffff00a0	\n \n \n \n	\n \n \n \n	\n \n \n \n	\n \n \n \n	\n \n \n \n	\n \n \n \n	\n \n \n \n	\n \n \n \n
0xffff00c0	\n \n \n \n	\n \n \n \n	\n \n \n \n	\n \n \n \n	\n \n \n \n	\n \n \n \n	\n \n \n \n	\n \n \n \n

Figure 4 data segment window for MMIO after running completed Lab4 assignment.

The input is achieved when the provided fileRead macro found in *lab4_testbench_rev##.asm* is executed. Your code does not need to do this.

For reference, from Figure 4, note where the first record in the given **data.csv** file, “**Kruger Industrial Smoothing,365\r\n**” is found. The location of **the letter ‘K’**(encoded in ASCII as byte 0x4b or 64 in base 10) **is at 0xffff0000**. The location of the character **digit ‘3’**, i.e. the start of the income,(encoded in ASCII as byte 0x33 or 51 in base 10) **is at 0xffff001c**. If you count the bytes in the string “Kruger Industrial Smoothing,365\r\n” with ‘K’ being the 0th character, then ‘3’ is the 28th character (0x1c), so this makes sense.

Assignment Project Exam Help

File Record Pointers

We need a systematic way to reference the memory locations where the **stock name** and **income** from the file buffer at 0xffff0000. Remember that the stock names and stock incomes can be both of varying lengths of characters. Thus, **we set aside the memory location 0x10040000 (heap) for a table containing the locations (addresses) of the first character appearing for each stock name and income respectively** (i.e. one for the name, one for the number) of each record in the CSV file.

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10040000	0xffff0000	0xffff001c	0xffff0021	0xffff002b	0xffff002e	0xffff0042	0xffff0047	0xffff005a
0x10040020	0xffff005f	0xffff0073	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Figure 5 data segment window for heap after running completed Lab4 assignment.

This is achieved when the testbench (*lab4_testbench_rev##.asm*) runs the provided function **allocate_file_record_pointers** with the **arguments in a0 and a1** registers being the file size in bytes (119 in our given example from *data.csv*) and the starting address of file buffer (0xffff0000 in our given example from *data.csv*), respectively. The *allocate_file_record_pointers* function is provided in the *allocate_file_record_pointers.asm* file separately through the .include statement in *lab4_testbench_rev##.asm*.

For reference, from Figure 5, note in our example **data.csv** the **first record's** location of start of **income name (0xffff0000)** and **income value(0xffff001c)** are stored as words in consecutive heap memory locations **0x10040000** and **0x10040004** respectively. Likewise, the **second record's** (i.e. “**Kramerica,0\r\n**”) location of start of **income name (0xffff0021)** and **income value(0xffff002b)** are stored as words in consecutive heap memory locations **0x10040008** and **0x1004000c** respectively. And so on and so forth. **It is left as a HIGHLY recommended exercise that the student verifies this pattern for the remaining records in the CSV file and how they are allocated in the memory locations as shown in Figure 5.** As we see in Figure 5, the 10 non zero heap memory locations from 0x10040000 to 0x10040024 indicate there were originally $10/2 = 5$ records in our *data.csv* file. This value (no. of records) is returned in a0.

Student coded functions

All student coded functions are to be written in the .asm files listed in the *lab4_testbench_rev##.asm* file using the .include statement (excluding *allocate_file_record_pointers.asm*). **The functions written MUST abide by the register saving conventions of RISC-V.**

1. **length_of_file(a1)** - This function is to be written in *length_of_file.asm*. It accepts as argument in a1 register the buffer address holding file data and returns in a0 the length of the file data in bytes.

From our example involving *data.csv*, `length_of_file(0xffff0000)=119`

2. **income_from_record(a0)** - This function is to be written in *income_from_record.asm*. It accepts as argument in a0 register the pointer to start of numerical income in a record. It returns the income's numerical value in a0.

From our example involving *data.csv*, `income_from_record(0x10040004)=365`. This is the income from the stock of Kruger Industrial Smoothing. `income_from_record(0x1004000c)= 0`. This is the income from the stock of KramERICA.

You may use the **mul** instruction if you feel your student code involves the multiplication operation.

3. **totalIncome(a0,a1)** - This function is to be written in *totalIncome.asm*. a0 contains the file record pointer array location (0x10040000 in our example) But your code MUST handle any address value. a1 contains the number of records in the CSV file. a0 then returns the total income (add up all the record incomes).

From our initial example *data.csv*, the call to `totalIncome(0x10040000, 5)` will return 1007

4. **maxIncome(a0,a1)** - This function is to be written in *maxIncome.asm*. a0 contains the file record pointer array location (0x10040000 in our example) But your code MUST handle any address value. a1 contains the number of records in the CSV file. a0 then returns the heap memory pointer to the actual location of the record stock name in the file buffer.

From our example involving *data.csv*, `maxIncome(0x10040000, 5)= 0x10040010`. Observe from Figure 5 that this address value points to the location of the stock name "Vandelay Industries", which is valued at the maximum value of \$500, and [proving that even in times of market uncertainty , the latex polymer industry is booming as ever.](#)

5. **minIncome(a0,a1)** - This function is to be written in *minIncome.asm*. a0 contains the file record pointer array location (0x10040000 in our example) But your code MUST handle any address value. a1 contains the number of records in the CSV file. a0 then returns the heap memory pointer to the actual location of the record stock name in the file buffer.

From our example involving *data.csv*, `maxIncome(0x10040000, 5)= 0x10040008`. Observe from Figure 5 that this address value points to the location of the stock name "KramERICA", which is valued at the minimum value of \$0, [proving once and for all, a sales pitch about a "coffee table book about coffee tables" is an absolutely terrible idea.](#)

To keep the code simple for both maxIncome and minIncome functions, you may assume that no two entries in the CSV file will have the same income field value.

Test Cases

Your Lab4 Google Drive folder called TestCases contains more test data files: *data1.csv*, *data2.csv* and *data3.csv*, and *data0.csv*. To test with these you have to copy any one of them to *data.csv* in your working directory. The initial *data.csv* you use by default is the same as TestCases/*data0.csv*. It corresponds to the file described in this write-up.

Automation

Note that our grading script is automated, so it is imperative that your program's output matches the specification exactly. The output that deviates from the spec will end up with a poor score.

Files to be submitted to Gradescope

length_of_file.asm
income_from_record.asm
maxIncome.asm
minIncome.asm
totalIncome.asm

You will NOT be editing the following files, and you will NOT BE UPLOADING THEM:

lab4_testbench_rev##.asm
allocate_file_record_pointers.asm
macros_rev#.asm

DO NOT UPLOAD ANY DATA FILES. We use our own data files to test your program.

Assignment Project Exam Help

ORDER TO IMPLEMENT FUNCTIONS

First make sure you have set up RARS to work on 64-bit mode or programs won't work.

IMPORTANT: START OUT BY SUBMITTING THE FILES AS THEY ARE WITHOUT MODIFICATION. GRADESCOPE WILL GIVE YOU 20 POINTS AS IS GIVEN THAT THE PROGRAM ASSEMBLES PROPERLY WITHOUT MODIFICATION. DO THIS AS SOON AS POSSIBLE!! YOU MUST SUBMIT ALL THE NEEDED FILES AS DESCRIBED ABOVE OR THE PROGRAM WON'T ASSEMBLE PROPERLY.

Make sure you read the code in the lab4_testbench_rev#.asm, and the file macros_rev#.asm you will get many ideas from reading these.

Submit using Gradebook. Do it as many times as you want, there is no penalty for multiple submissions. If your grade gets worse you can go back to your best prior grade.

*After you submit the files without any edits, you must get the **length_of_file** function working next in order to make any further progress. Using the default data.csv you should get a length of 119.*

After you get length_of_file working, you must get the income_from_record function working. Double check that the output displayed matches the input. income_from_record_ideas.asm will give you some coding ideas that will help. Once income_from_record works, get totalIncome, and maxIncome and minIncome functions working in that order. The last two are very similar.

If you get all that working go for the extra credit described next.

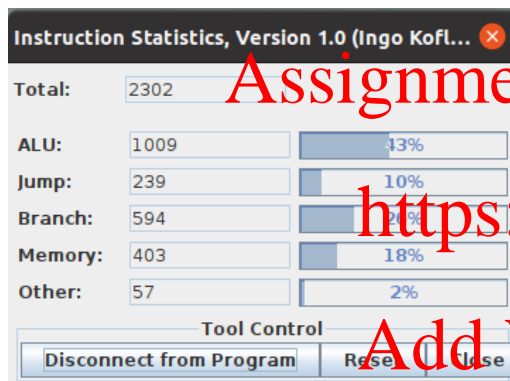
*This is a medium difficulty lab. So start early and work with the TA's and tutors to help you complete it. **DO NOT PROCRASTINATE.***

POTENTIAL SIZABLE EXTRA CREDIT

This is a great opportunity for you to learn assembly language, and become somewhat of a near expert in RISC-V programming, while you work on getting extra credit. With continuous regrading and some significant effort, you should be able to get a perfect score in this lab. If you do, I will be giving you the opportunity to get some extra credit, and learn more while you are at it.

The idea is that with some effort if you get 100 points on this lab, you can work on optimizing it, too. In particular you can use what you have learned about caller and callee saving conventions when using nested functions (which this lab does do). My most optimized implementation of this lab executed running a total of **2228** instructions.

You can learn how many instructions you executed by accessing the **Tools** menu then the **Instruction Statistics** function, connecting it to your program to it and running it. I put in a fair effort myself at using simple tricks to optimize the code, also following the guidelines for saving registers. The register spilling optimizations are described in the lecture notes. Sooo, if you can match **2300** or even beat my number you will **get an extra 20 percent extra credit** for Lab4 to help you improve your grade. Think of it. This is an opportunity to beat your teacher at his game. Not only will you get extra credit but also **great bragging rights** :) Worst case, even if you don't get the extra credit, you may have fun trying, and learn a thing or two. However, you have to complete the assignment first to perfection before you can try this.



Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

SOME TIPS

We will be discussing how to debug your code in class along with numerous tricks and techniques to make your life easier. You can use some of the macros on your code to print out intermediate results if you feel you need to. You can also put breakpoints in your code and use the powerful built-in debugger.

A caveat: **Be careful: use `lwu` not `lw` when loading a pointer from memory.** The `lw` instructions does sign-extension into 64-bits. 64-bit addresses that do not work in RARS and result in unfriendly run-time errors. RARS uses 32-bit pointers, and a 32-bit address space, even though the RV64I model has 64-bit registers. To create pointers it is best to use the unsigned version of these instructions.

A Note About Academic Integrity

This is the lab assignment where many students get flagged for cheating. Please review the pamphlet on [Academic Dishonesty](#) make sure you understand the difference between acceptable and unacceptable collaboration.

You can get advice and help learning how to do this, but you must do this assignment all by yourself. Write comments that shows you know what you are doing! COPYING CODE FROM SOMEONE ELSE OR THE INTERNET IS NOT ALLOWED.

Grading Rubric (100 points total + 20% extra credit)

The following rubric applies provided you have fulfilled all criteria in **Minimum Submission Requirements**. Failing any criteria listed in that section would result in an automatic grade of zero **which cannot be eligible for applying for a regrade request**.

20 pt `lab4_testbench_rev##.asm` **assembles** without errors (so even if you submit `lab4_testbench_rev##.asm` with all the other required .asm files **COMPLETELY** unmodified by you, you will get 20 pts).

30 pt `length_of_file.asm` works

20 pt `income_from_record.asm` works

10 pt `totalIncome.asm` works

10 pt `maxIncome.asm` works

10 pt `minIncome.asm` works

AGAIN: Execute this program perfectly using 2300 instructions of execution or less, and you will get an extra 20% extra credit for this lab!! Note: efficient register spilling will help you with this. This is a good way to make up for lost points.

Regrade Requests

There will not be any regrades for this Lab. Gradescope gives you the ability to see your score, and it is your responsibility to fix any grading issues prior to the deadline by working with the instructor, TA's and tutors.

<https://powcoder.com>

Legal Notice

All course materials and relevant files located in the Lab4 folder in the course Google Drive must not be shared by the students outside of the course curriculum on any type of public domain site or for financial gain. Thus, if any of the Lab4 documents is found in any type of publicly available site (e.g., GitHub, stack Exchange), or for monetary gain (e.g., Chegg), then the original poster will be cited for misusing CSE12 course-based content and will be reported to UCSC for academic dishonesty.

In the case of sites such as Chegg.com, we have been able to locate course material shared by a previous quarter student. Chegg cooperated with us by providing the student's contact details, which was sufficient proof of the student's misconduct leading to an automatic failing grade in the course.