CSE12 Lab 4: Simple CSV File Analysis

Due March 15th 2023 , 11:59 PM

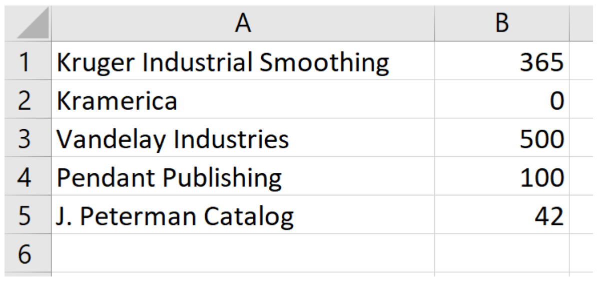
Minimum Gradescope Submission Requirements

Ensure that your Lab4 submission contains the following files:

- *lab4w23_testbench.asm*
- *allocate_file_record_pointers.asm*
- *income_from_record.asm*
- *length_of_file.asm*
- *maxIncome.asm*
- *minIncome.asm*
- *totalIncome.asm*
- *data.csv*

Objective

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 in 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:



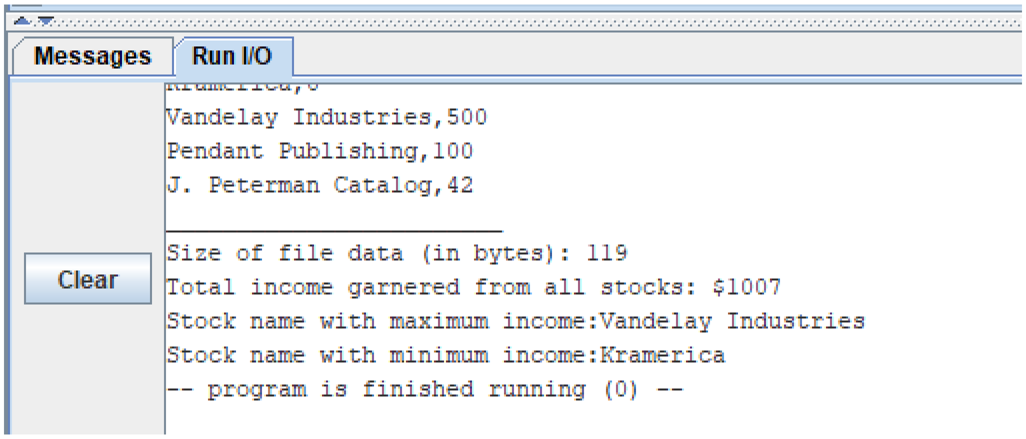
	A	B
1	Kruger Industrial Smoothing	365
2	Kramerica	0
3	Vandelay Industries	500
4	Pendant Publishing	100
5	J. Peterman Catalog	42
6		

Figure 1 data.csv file in Excel

This file shows the stock returns from an investment portfolio over a year. "A" column contains the stock name and "B" column indicates the returns in USD (***We will assume that there is no negative stock return in all our CSV files***). Then running the *lab4w23_testbench.asm* file in RARS that takes *data.csv* as the input CSV file should yield the following analysis:

1. Find the total file size in bytes (excluding any metadata generated by your OS)
2. Calculate the total income generated from all stocks
3. Find the stock that gave maximum returns and the one with minimum returns..

When we run (***the completed***) *lab4w23_testbench.asm* in RARS, we should get the output console as shown below.



```
Kramerica,0
Vandelay Industries,500
Pendant Publishing,100
J. Peterman Catalog,42
_______________________
Size of file data (in bytes): 119
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) --
```

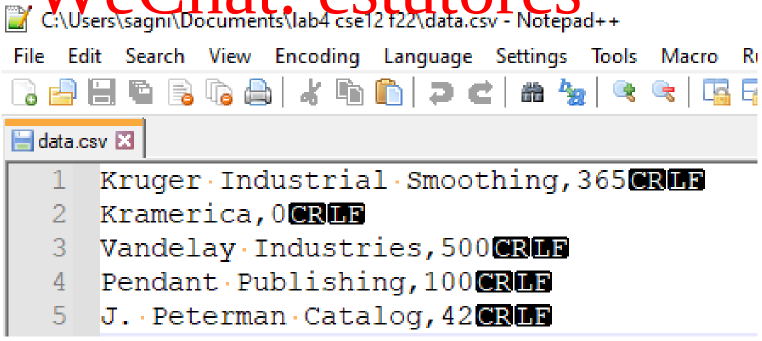
Figure 2 After running the lab4w23_testbench.asm file with the Lab4 assignment fully completed

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 both a <CR> followed by 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 get the following view showing the placement of the carriage return and line feed characters. .


C:\Users\sagni\Documents\lab4 cse12 f22\data.csv - Notepad++
File Edit Search View Encoding Language Settings Tools Macro
data.csv

```
Kruger·Industrial·Smoothing,365CRLF
Kramerica,0CRLF
Vandelay·Industries,500CRLF
Pendant·Publishing,100CRLF
J.·Peterman·Catalog,42CRLF
```

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.

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 <u>immediately</u> by the stock price expressed as an <u>unsigned</u> 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 black spaces. This assumption will make the CSV file analysis by RARS a bit 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 Google drive, you will see 7 assembly files. They are meant to read *(and understood*) **in sequence**:

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. *RISC-V_function_convention.pdf* – This pdf describes the conventions of how we use the various 32 RV64I registers in terms of creating a program with multiple functions. The documentation is provided such that students can start working on implementing the required function for Lab4. Studying the comments in add_function.asm alongside reading this pdf should be sufficient to create the required functions for this assignment.

4. *lab4w23_testbench.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 correctly generated output as far as using the function *allocate_file_record_pointers* from *allocate_file_record_pointers.asm* is concerned.

5. *allocate_file_record_pointers.asm* - This .asm file contains a function that creates an internal reference table in memory that points to the location of the start of a string indicating stock name and the start of the stock price for each and every record/entry in the data.csv file. This function has been fully written out for you. ***You must not edit or modify this file.***

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. *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. Refer to Figure 2 for an example.

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 these files and make sure to open them in **RARS Text editor only**. Else the comments and other important code sections won't be properly highlighted and can be a hindrance to learning assembly language intuitively.

These 9 files have enough comments in the source code to jumpstart 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, in the ppt files which have the title slide "Von Neuman and RISC- V". These slides are very self-explanatory and it is encouraged you start reading them even if the instructor hasn't started discussing them in lecture.***

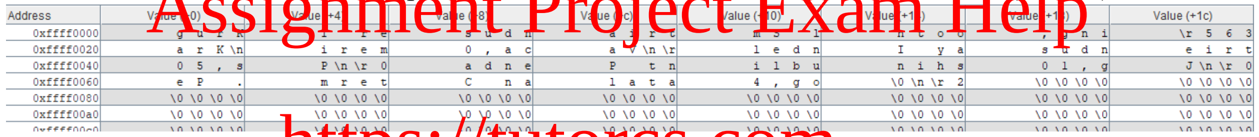
For the usage of macros (which are utilized heavily in this lab to generate ecalls), please also refer to the RARS documentation on macros and ecalls as well.

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 an input and saved to the file buffer location 0xffff0000 (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 1 , g	J \n \r 0
0xffff0060	e P .	m r e t	C n a	l a t a	4 , g o	\0 \n \r 2	\0 \0 \0 \0	\0 \0 \0 \0
0xffff0080	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0xffff00a0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0xffff00c0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0

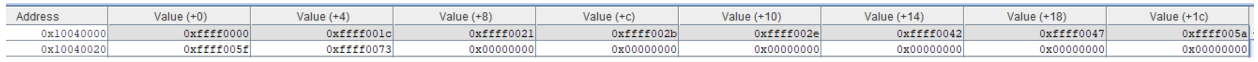
Figure 4 data segment window for MMIO after running completed Lab4 assignment.

This is achieved by running the provided fileRead macro in *lab4w23_testbench.asm* .

For reference, from Figure 4, note the first record in the given *data.csv* file, "Kruger Industrial Smoothing,365\r\n". The location of the letter 'K'(encoded in ASCI as byte 0x4b or 64 in base 10) is 0xffff0000. The location of the character digit '3', i.e. the start of the income,(encoded in ASCI as byte 0x33 or 51 in base 10) is 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.

File Record Pointers

We need a systematic way to reference the memory locations of both 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 of the first character appearing in the stock name and income respectively 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 by running the provided function *allocate_file_record_pointers* in *lab4w23_testbench.asm* 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(0x0ffff0000 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 *lab4w23_testbench.asm.*

For reference, from Figure 5, note in *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 *lab4w23_testbench.asm* file using the .include statement (excluding *allocate_file_record_pointers.asm*). ***The functions written MUST abide by the register saving conventions as outlined in RISC-V_function_convention.pdf.***

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 example involving *data.csv*, totalIncome(0x10040000, 5)= 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

The Lab4 folder in Google Drive contains some test cases in the testCases subfolder for the case when filePath is changed to *data2.csv* and *data3.csv* respectively in *lab4w23_testbench.asm*. The corresponding output that was obtained for these two files from running *lab4w23_testbench.asm* for them is provided in this subfolder as well.

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 cause point deduction.

Files to be submitted to your Lab4 Gradescope submission

lab4w23_testbench.asm

allocate_file_record_pointers.asm ***(DO NOT EDIT/MODIFY THIS FILE)***

income_from_record.asm

length_of_file.asm

maxIncome.asm

minIncome.asm

totalIncome.asm

data.csv ***(DO NOT EDIT/MODIFY THIS FILE)***

A Note About Academic Integrity

This is the lab assignment where most students continue to get flagged for cheating. Please review the pamphlet on Academic Dishonesty and look at the examples in the first lecture for acceptable and unacceptable collaboration.

You should be doing this assignment completely all by yourself!

Grading Rubric (100 points total)

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 *lab4w23_testbench.asm* ***assembles*** without errors (so *even if you submit lab4w23_testbench.asm with all the other required .asm files COMPLETELY unmodified by you, you would still get 20 pts!*)

80 pt output in file *lab3_output.txt* matches the specification:

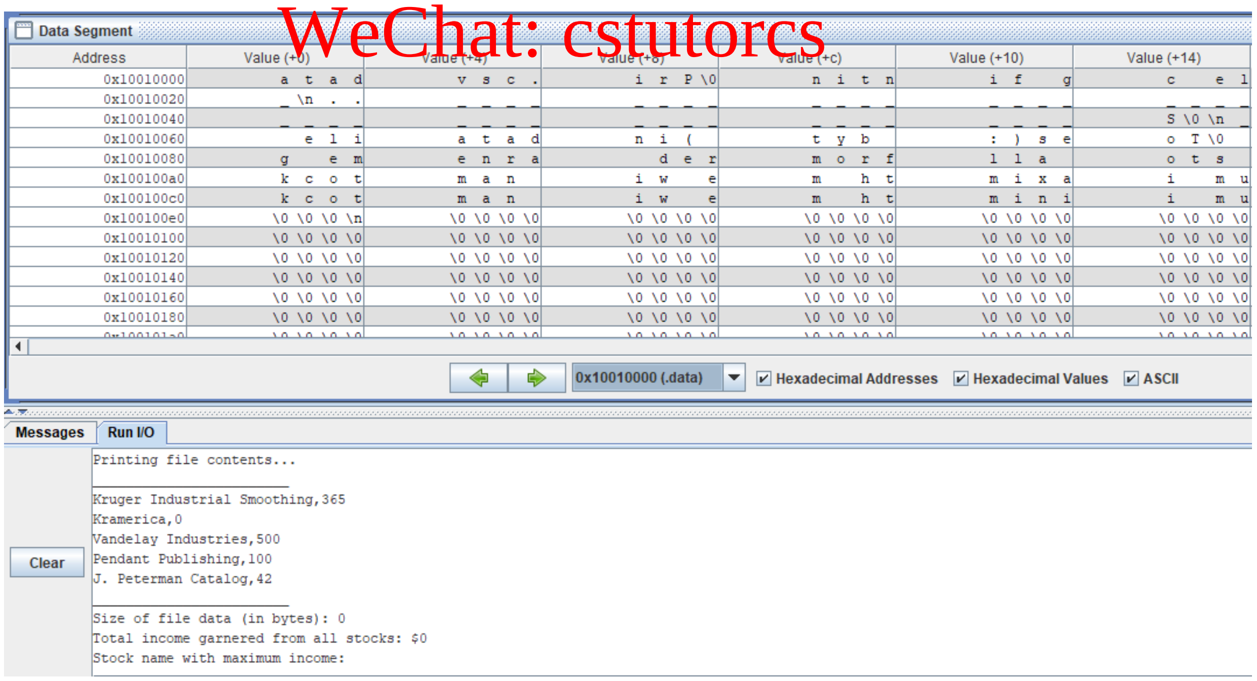
- **30 pt** *length_of_file.asm* works
- **20 pt** *income_from_record.asm* works
- **10 pt** *totalIncome.asm* works
- **10 pt** *maxIncome.asm* works
- **10 pt** *minIncome.asm* works

Remember that you will get the minimum 20 pts only if the testbench assembles correctly. There is no need to then run it for the minimum points.

The testbench, with the other .asm files left as it is, is supposed to give an error when it tries to access either the maxIncome or minIncome functions. You need to correctly implement those two before you see a fully working testbench.

Note however that even when you ***do*** run the given testbench, you should still see some deliverable output.

For example, here's the I/O console when I run the testbench as is. You will see that every code before line 126 9as you exit maxIncome function) has been executed correctly:



Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)
0x10010000	a t a d	v s c .	i r P \0	n i t n	i f g	c e l
0x10010020	_ \n . .	_ _ _ _	_ _ _ _	_ _ _ _	_ _ _ _	_ _ _ _
0x10010040	_ _ _ _	_ _ _ _	_ _ _ _	_ _ _ _	_ _ _ _	S \0 \n _
0x10010060	e l i	a t a d	n i (	t y b	:) s e	o T \0
0x10010080	g e m	e n r a	d e r	m o r f	l l a	o t s
0x100100a0	k c o t	m a n	i w e	m h t	m i x a	i m u
0x100100c0	k c o t	m a n	i w e	m h t	m i n i	i m u
0x100100e0	\0 \0 \0 \n	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010100	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010120	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010140	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010160	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010180	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0

0x10010000 (.data) ✓ Hexadecimal Addresses ✓ Hexadecimal Values ✓ ASCII

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): 0
Total income garnered from all stocks: $0
Stock name with maximum income:
```

Figure 6 Running the testbench without modifying any of the starter code files results in partial output delivered but enough to give you a starting idea!

Regrade Request Form

Click here for the Regrade Request Form. Only those regrade requests that were submitted ***within 1 week*** of having received your Lab 3 grade (an announcement would be made on Piazza) will be considered. Else it will be ignored.

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 Lab3 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.