Developing Soft and Parallel Programming Skills Using Project- Based Learning

TheRaspberryFive
Jay Patel, Marcus Aqui, Bonnie Atelsek, Jacqes Agbenu, Zach Benator

Project-A3
Semester (Spring-2019)

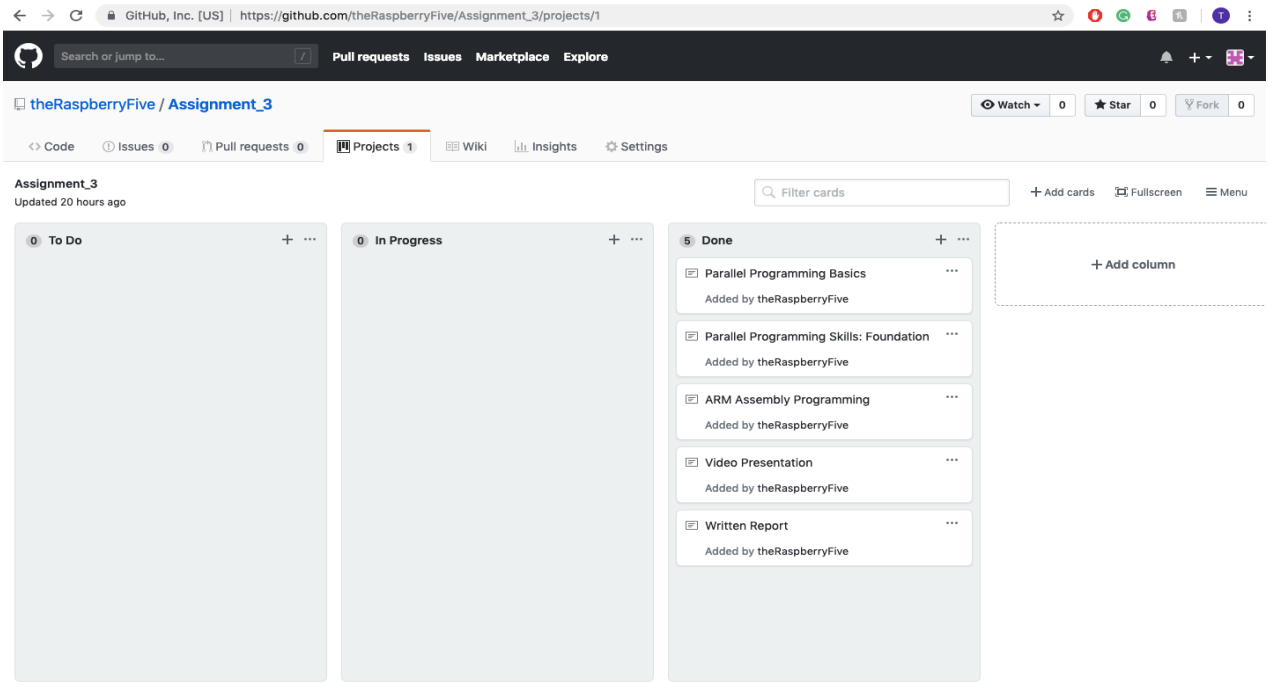
Task 1

Planning and Scheduling:

Assignee Name	Email	Task	Duration (hours)	Due Date	Notes
Marcus Aqui	maqui1@student.gsu.edu	YouTube video edit and upload, Parallel programming (Part 1)	3 hrs	3/8	Make sure video is within time frame
Jay Patel (**Coordinator**)	jpatel118@student.gsu.edu	Set up Slack, create and write report, ARM programming	4 Hrs	3/8	Remember to print report and submit
Bonnie Atelsek	batelsek1@student.gsu.edu	Parallel programming (Part 1 and Part 2)	5 hrs	3/8	Take screenshot
Jacqes Agbenu	jagbenu1@student.gsu.edu	ARM programming	7 Hrs	3/8	Take Screenshot
Zach Benator	zbenator1@student.gsu.edu	GitHub, Parallel programming (Part 1)	3 hrs	3/8	Collect data and manage the files

Task 2

GitHub



Task 3

A) Foundation:

Define the following: Task, Pipelining, Shared Memory, Communications, Synchronization.

- Task - A program that is executed by a processor
- Pipelining - Breaking down a task in a way so that the output of one task is the input of another
- Shared Memory - Computer architecture where the processors share a bus and all tasks have access to the same memory
- Communications - The sharing of data between different tasks working in parallel
- Synchronization - Coordination between the parallel tasks in order to execute them in real time

Classify parallel computers based on Flynn's taxonomy. Briefly describe every one of them.

- SISD - Non-parallel computers that use one instruction stream and one data stream per clock cycle
- SIMD - A type of parallel computer which uses all processing units to execute the same instruction at any clock cycle. Each processing unit can work on a different element of data.
- MISD - A type of parallel computer which allows each processing unit to operate on data independently through separate instruction streams. A single data item is fed into multiple processing units
- MIMD - A type of parallel computer that allows each processor to execute a different instruction stream. Each processor can also work on a different data stream. This is the most common type of parallel computer
- Shared Memory, Threads, Distributed Memory/Message Passing, Data Parallel, Hybrid, SPMD, MPMD

What are the Parallel Programming Models?

- Shared Memory, Threads, Distributed Memory/Message Passing, Data Parallel, Hybrid, SPMD, MPMD

List and briefly describe the types of Parallel Computer Memory Architectures. What type is used by OpenMP and why?

- Shared Memory: UMA and NUMA. UMA stands for Uniform Memory Access, and NUMA stands for Non-Uniform Memory Access. In UMA, all processors have equal access to the memory, while in NUMA processors do not have equal access time when accessing the memory. OpenMP uses a shared memory architecture, which seems to have the capability of being either UMA or NUMA, which is probably good for versatility.

Compare Shared Memory Model with Threads Model?

- In threaded shared memory, processes are decomposed into simpler processes called threads that act as subroutines. The threads all connect to a common shared memory. In non-threaded memory, tasks share a common address space. Access to the memory is controlled by locks and semaphores, which threaded memory does not have. This management of memory can lead to difficulties with data locality, which does not seem like a problem for threaded memory, which uses a global memory.

What is Parallel Programming?

- Parallel Programming is a method of programming that makes use of multiple processors and data streams to complete tasks faster and more efficiently.

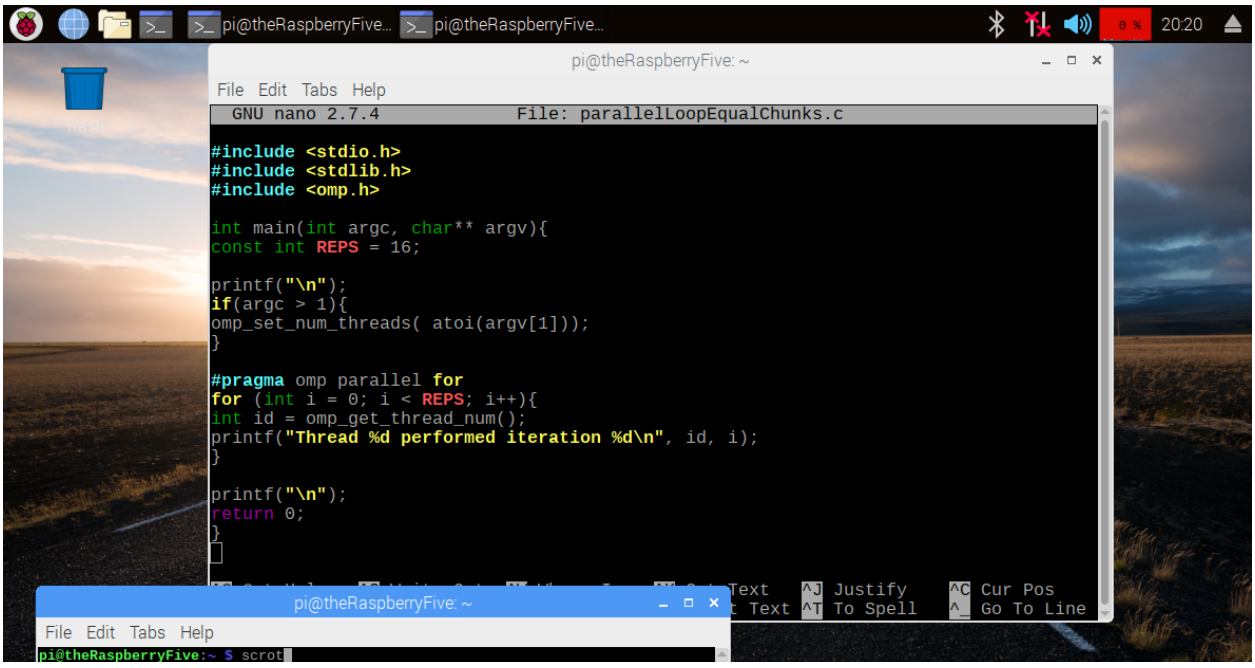
What is system on chip (SoC)? Does Raspberry PI use system on SoC?

- System on chip (SoC) is composed of the CPU, RAM, and GPU all squeezed onto one chip. Raspberry Pi does use SoC.

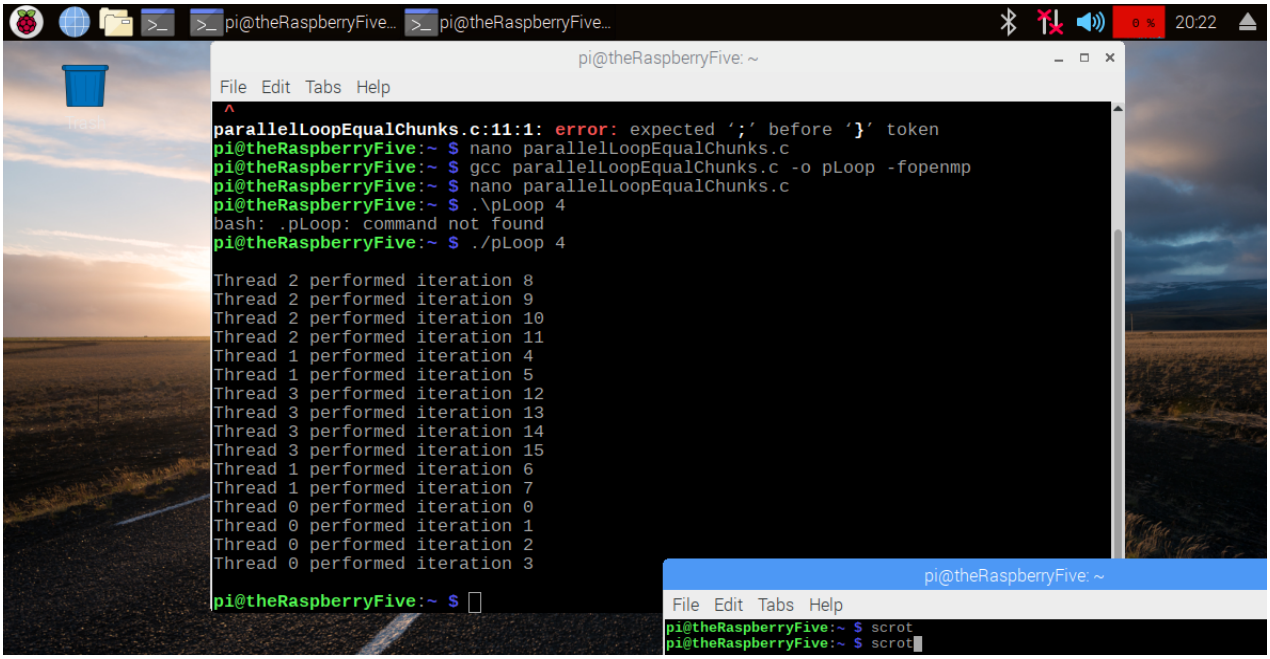
Explain what the advantages are of having a System on a Chip rather than separate CPU, GPU and RAM components.

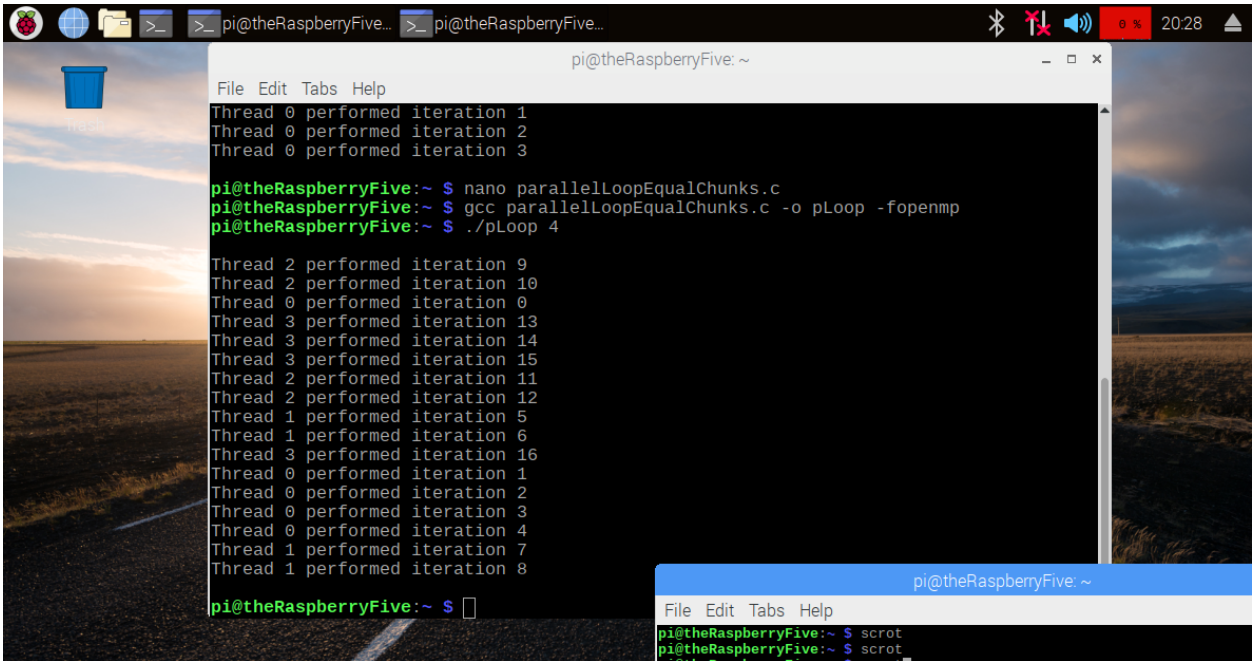
- The main advantage that the SoC has over the CPU, RAM, and GPU, in separate components, is that the SoC is much smaller than the other separate components. The size of the SoC allows the inclusion of complete computers in small devices such as smartphones and tablets.

B) Parallel Programming

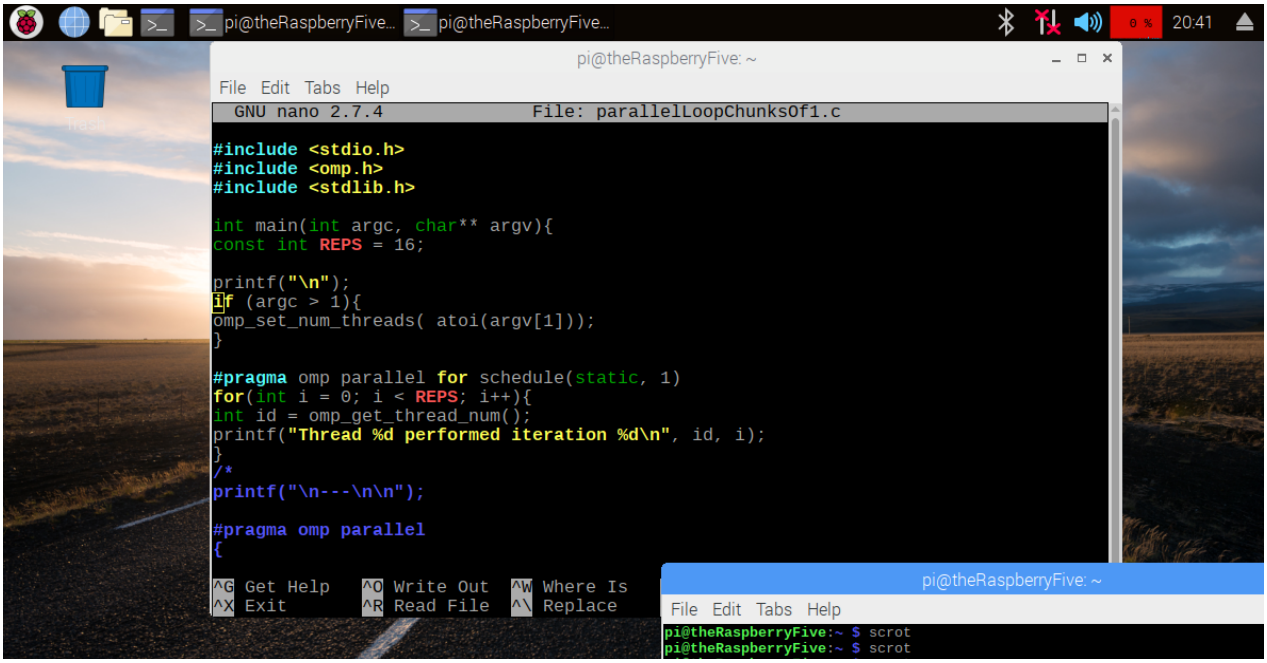


The first section of the assignment, the portion dealing with the parallelLoopEqualChunks.c file, was very simple. I copied, edited, and compiled it with no trouble. It ran according to how I thought it would, similar to the parallel programming we have done in the previous project, with forking a specific task, and calling a specific number of threads (4) when running the code. Verifying that the code ran as it should was also simple, I just checked that all the threaded results corresponded with the numerical sections they should have. Substituting other numbers for the amount of threads was something that I initially thought would cause a problem, but the code ran fine, it just had a different number of results and it divided the work unevenly when the number of iterations was not evenly divisible by the number of threads.

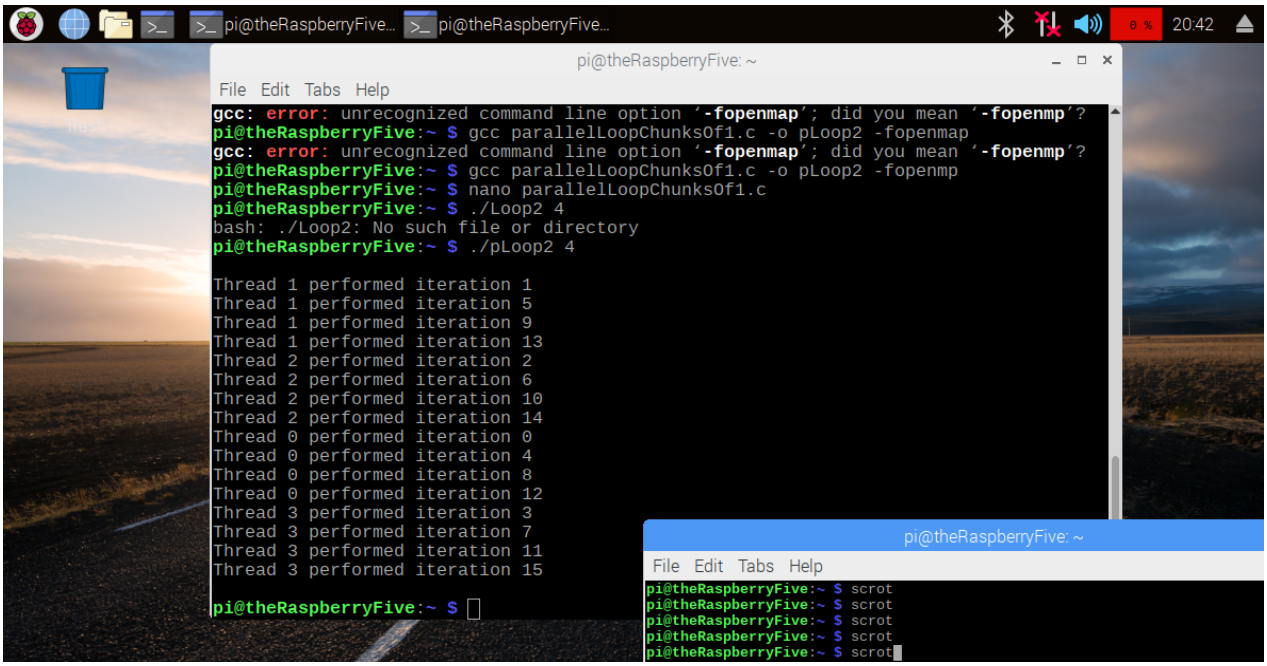


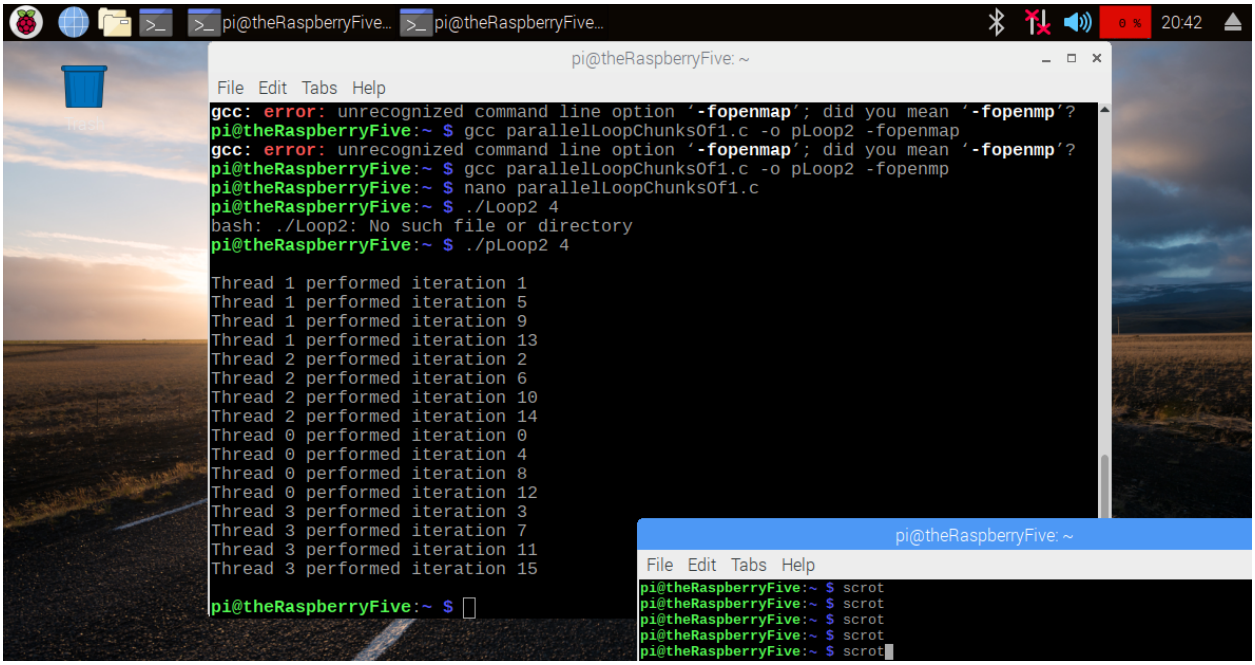


The next section, parallelLoopChunksOf1.c, was slightly more complicated for me to understand, but it worked fine. I had no trouble copying or running it, it seemed to function similarly to the previous code.

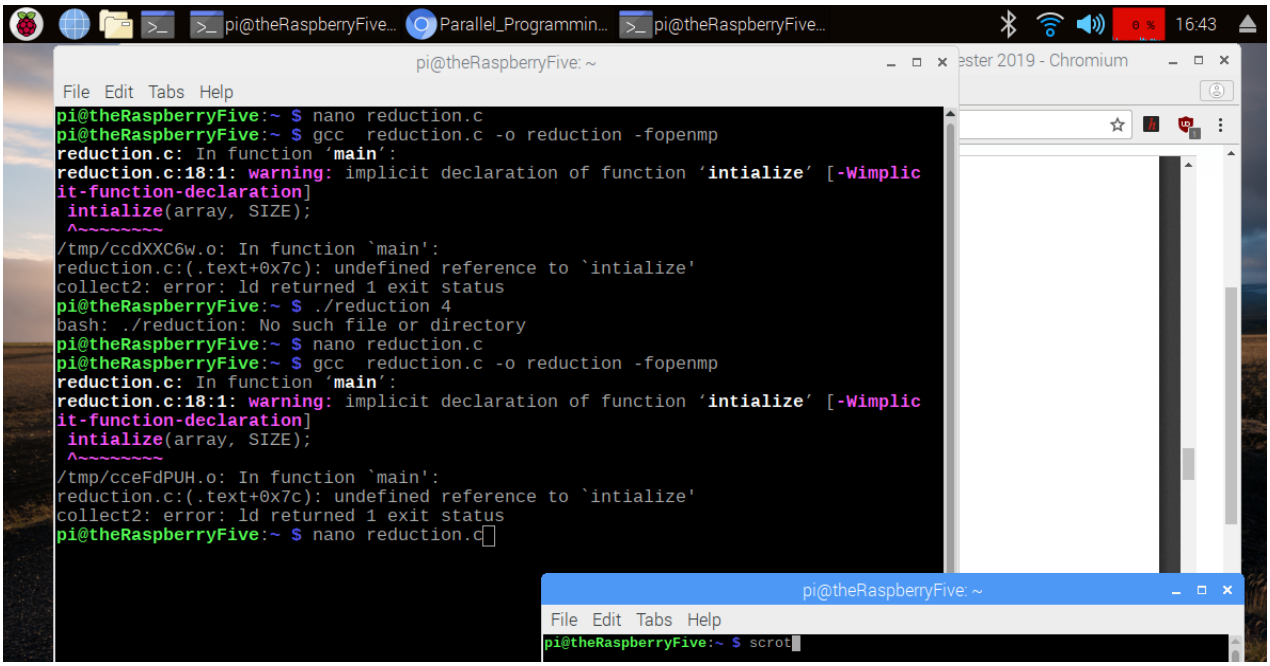


The output was similar to what we saw last time we ran code in parallel in the previous project, with the code outputting lines of the thread number and the iteration performed by said thread. There was one line of output for whatever the REPS variable was set as, and they were divided among threads in the same way as the above parallelLoopEqualChunks.c file was, with thread numbers handling specific chunks of information

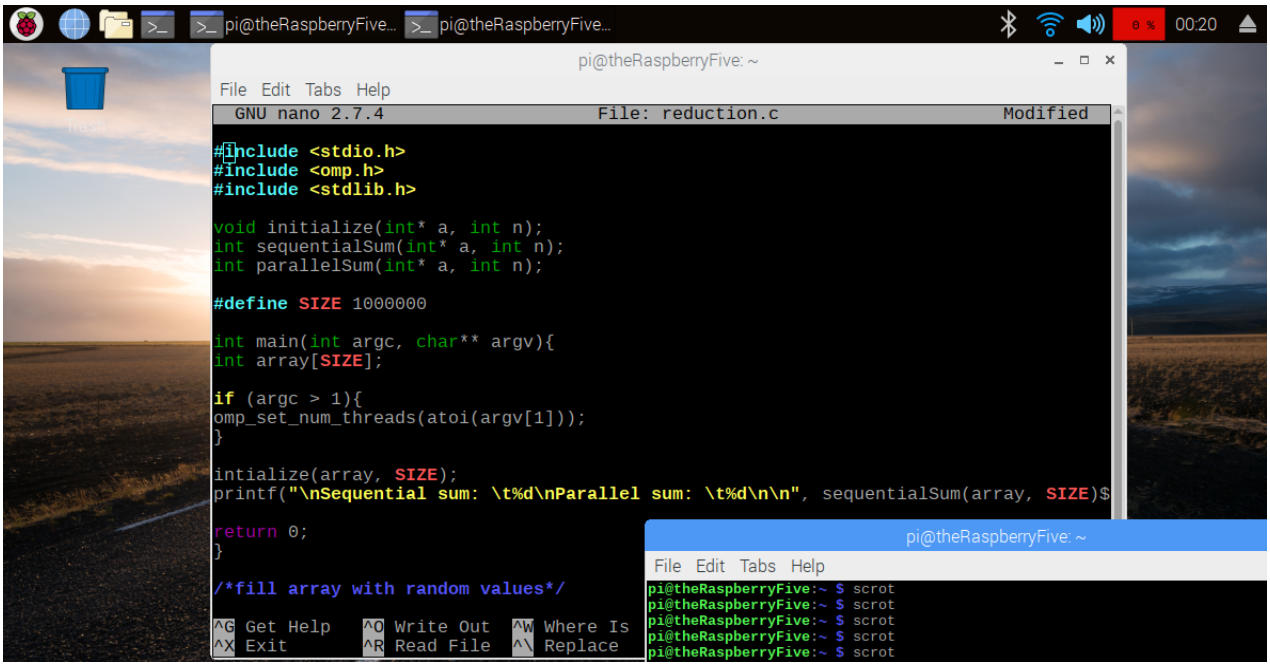


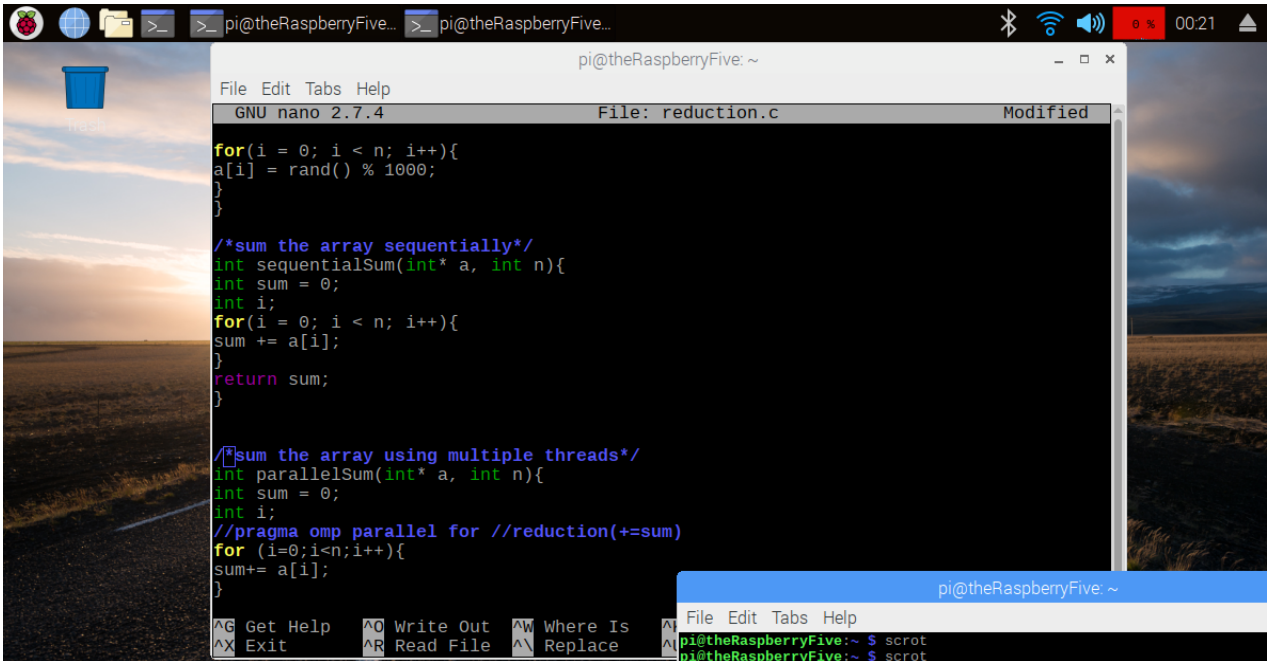


The fourth part was very difficult for me. I copied the code and tried to compile it, but I kept getting an error message that said that the initialize command on line 13 was being invoked statically and that could not be done. As a result, my code would not compile and I therefore could not run it. This problem made it impossible for me to move forward for a little while. I went over my code line by line to check if all the text was the same, and it seemed to be. I googled the issue to see if there was a work around, and even reread the document to see if that error was supposed to happen. However, since I could not find any help those ways, I looked up the same code used in the document online and copy and pasted it, thinking that I had probably written something wrong and not realized it.



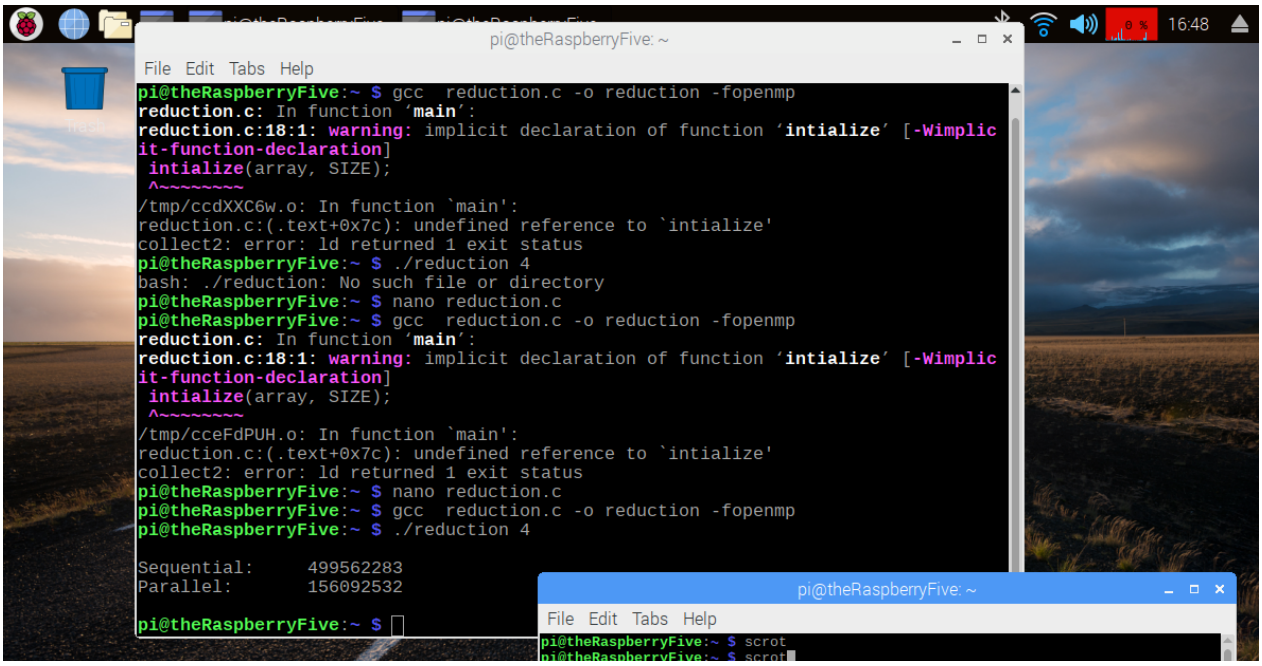
After doing this, my code compiled and ran with no problems. I looked over again to try and find where I went wrong the first time but I could not tell.



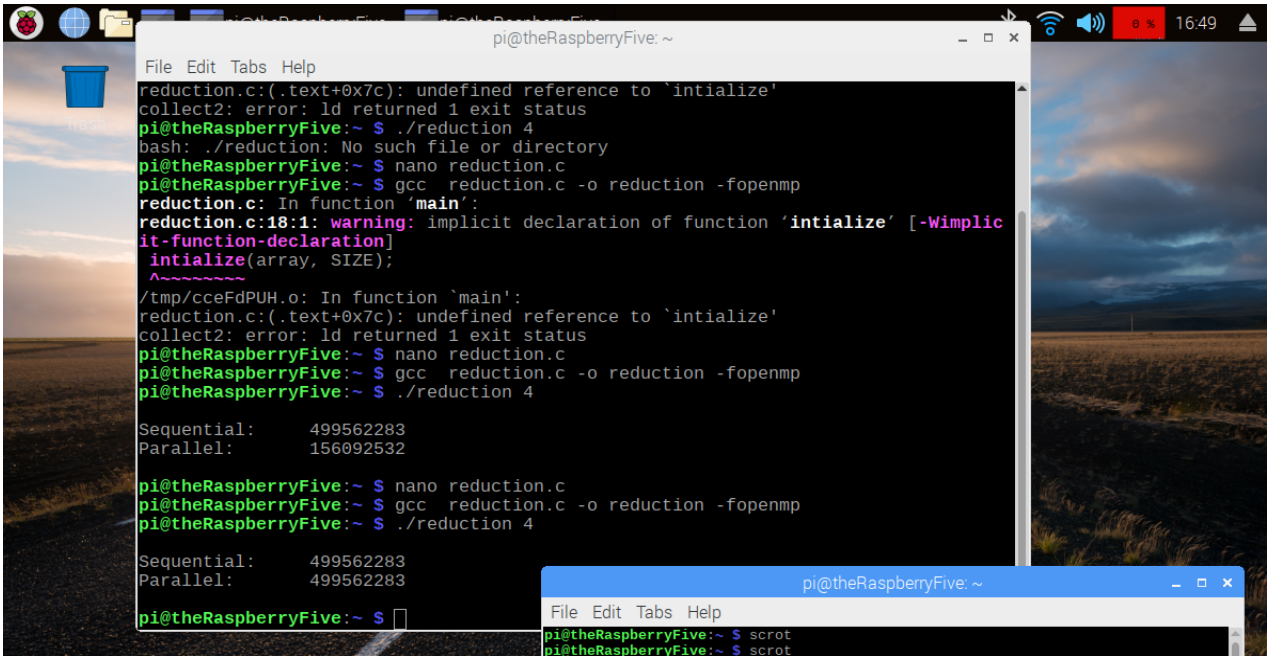


The output showed the parallel sum and the sequential sum next to each other, first when the parallelSum method was running incorrectly, and then after the code was modified, when the parallelSum method was running correctly.

At first, when running the code with the part in line 39 commented out, the parallelSum and sequentialSum values were not the same.



However, after modifying the code to uncomment line 39, compiling, and running in parallel, the results of parallelSum and sequentialSum were the same.

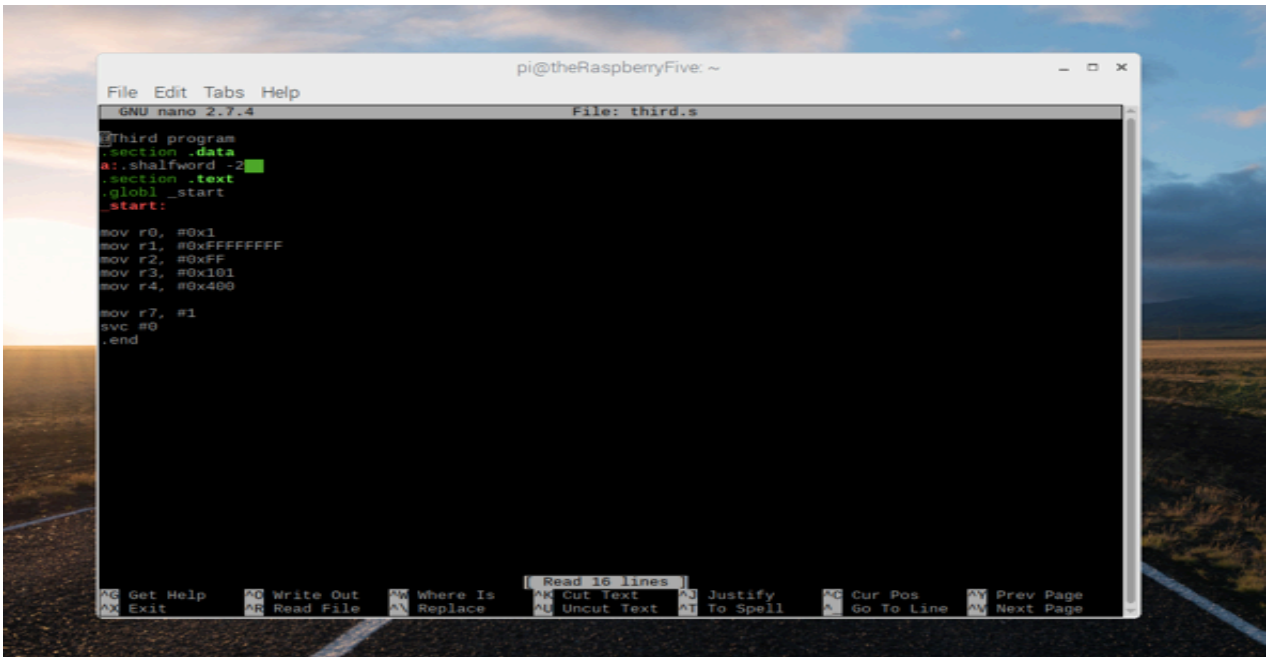


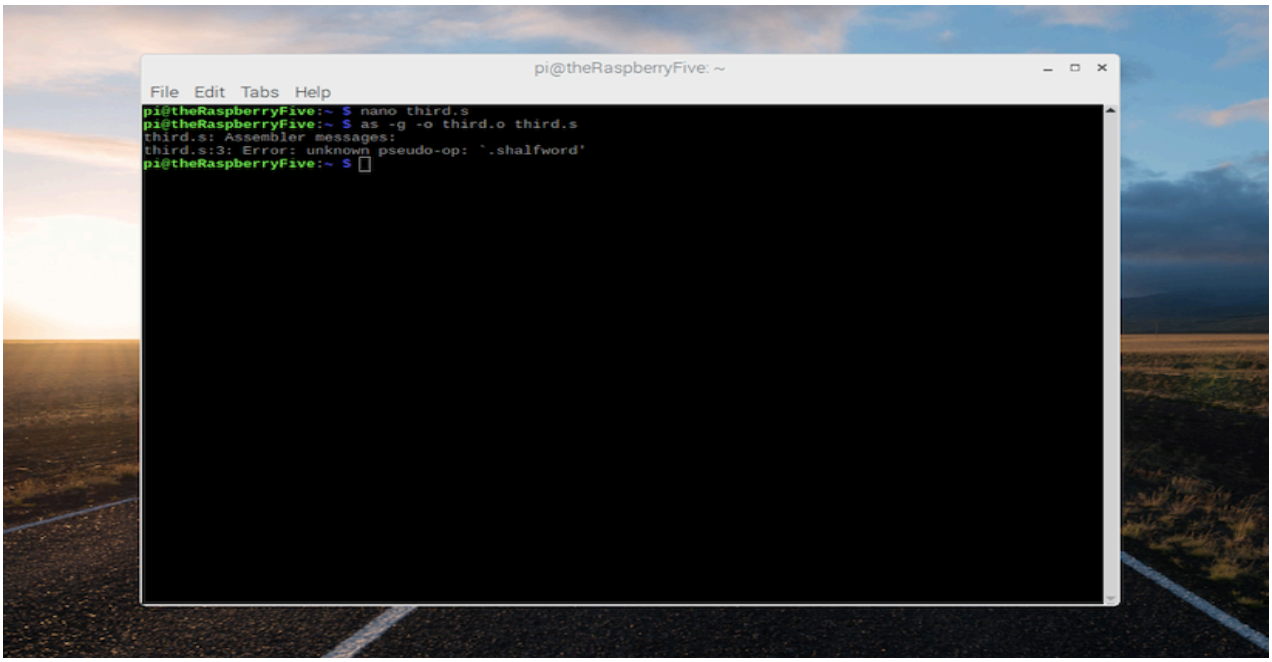
I am not entirely sure why the parallel without the reduction clause would run incorrectly; maybe it was counting wrong because it was running at the same time and double counting? The document says that it occurred because the variables were not private to each thread, which mostly aligns with what I thought.

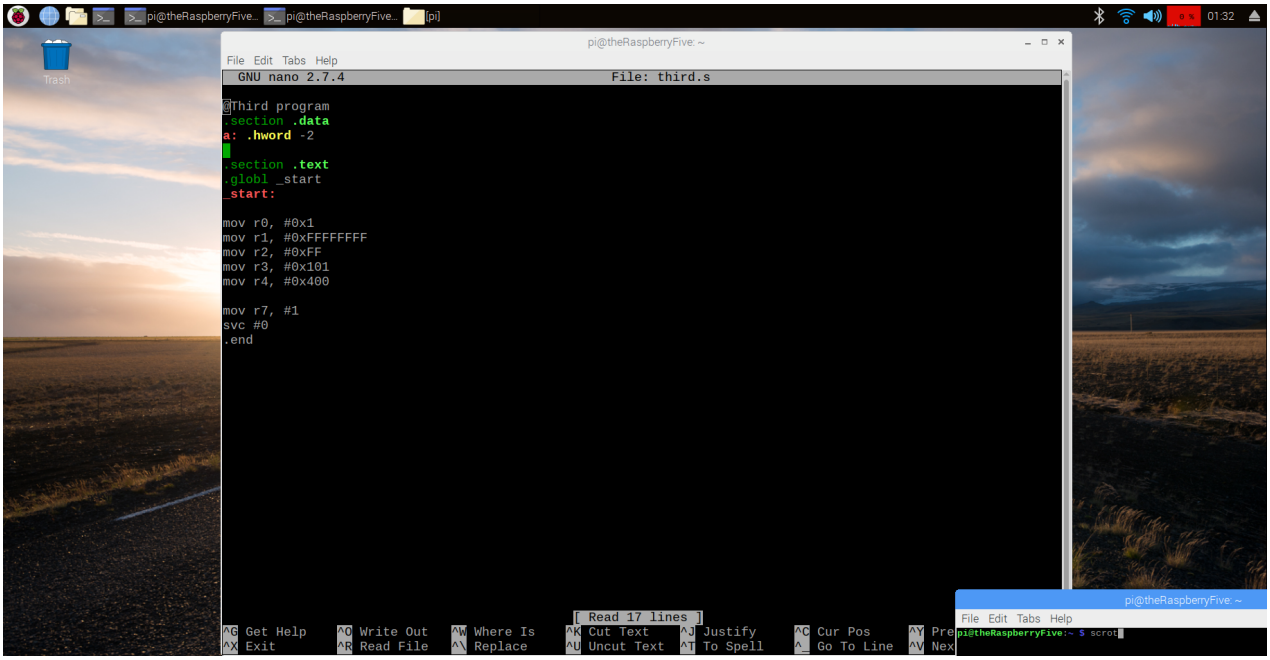
Task 4

A) ARM Programming

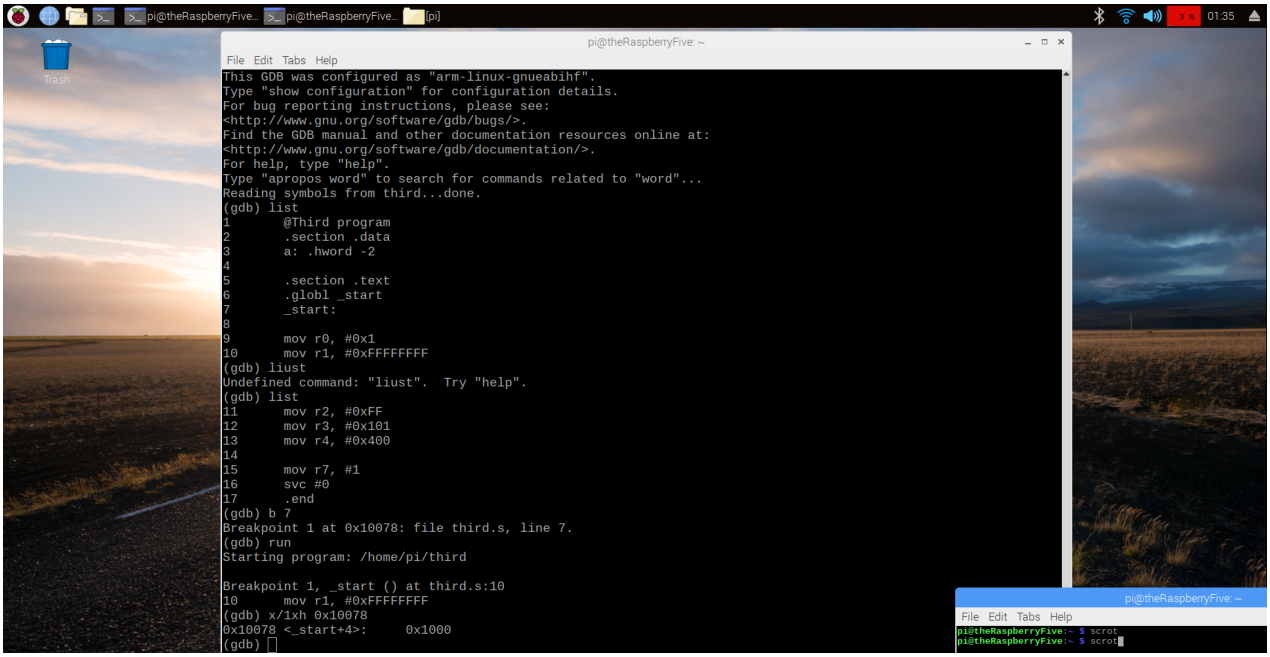
The initial part of the ARM programming was not that hard to finish. The error was confusing to fix since there is no signed halfword in ARM Programming.





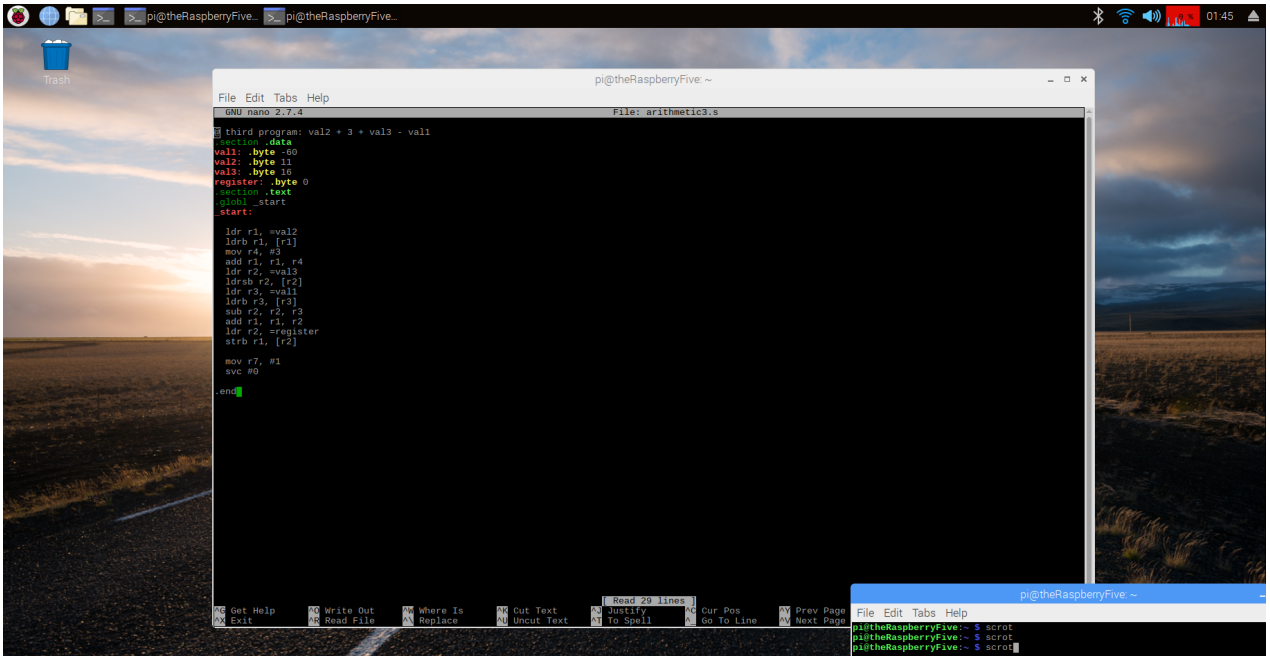


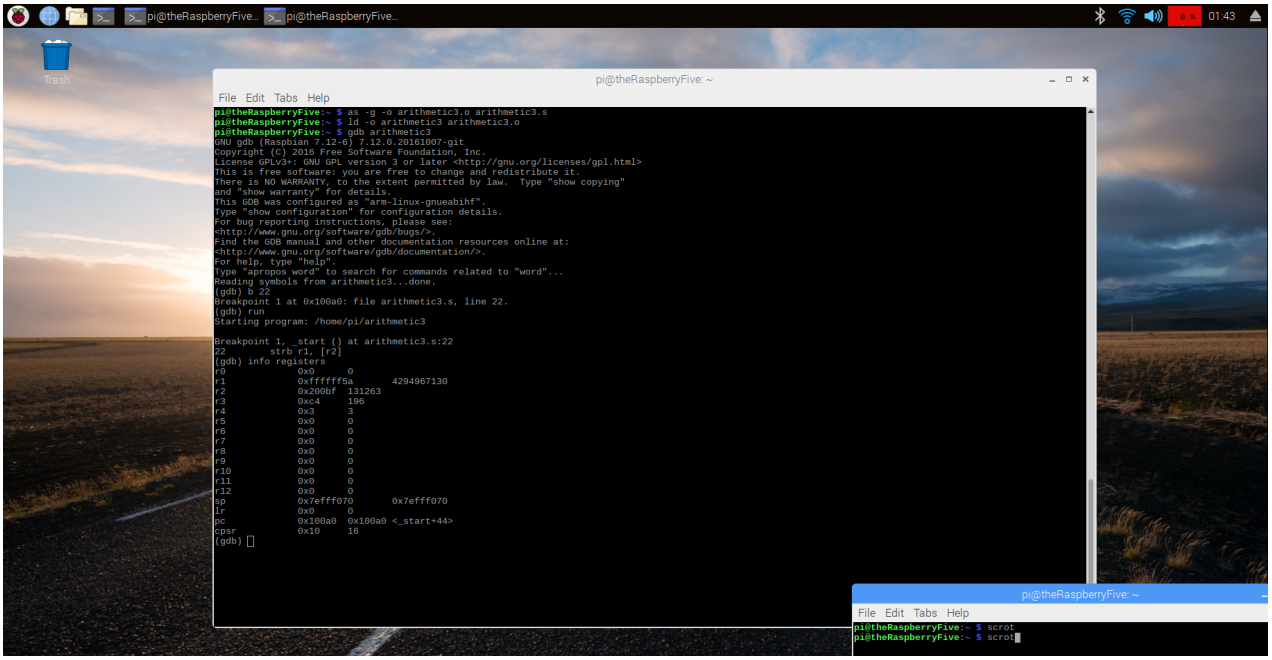
Printing out the hex value at a certain point in the was not too hard in this part of the programming.



B) ARM Programming

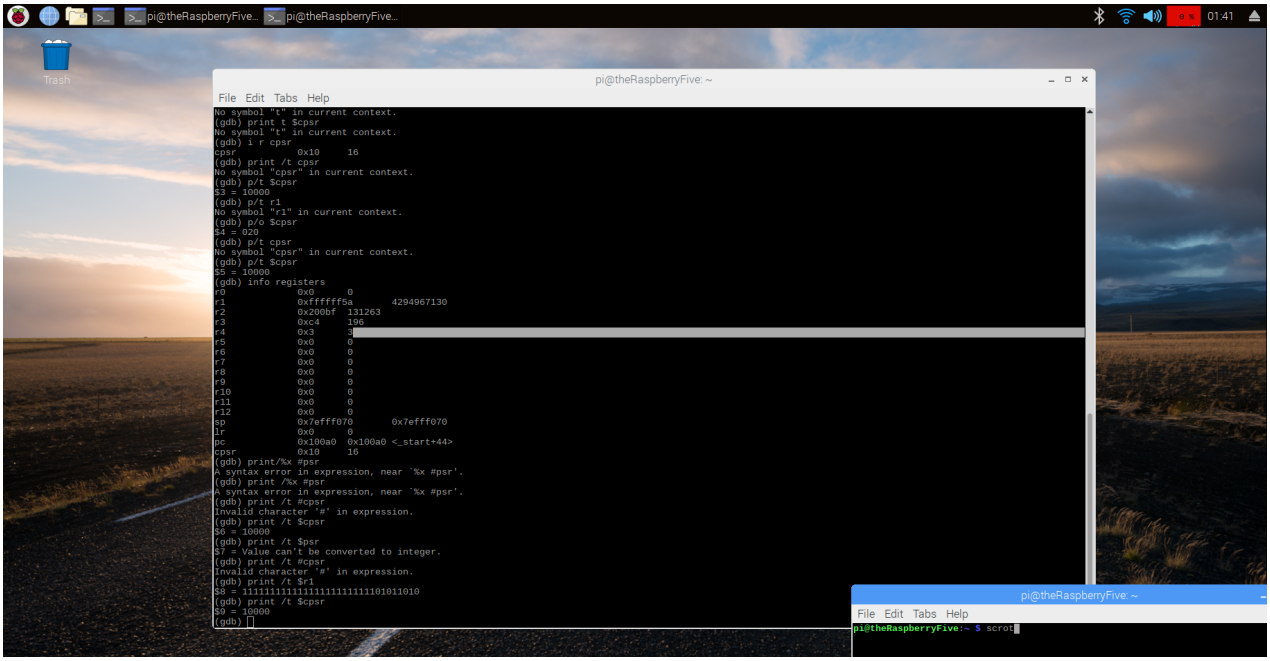
The third arithmetic section of this programming was trip from beginning to end. It was weird to code as a negative number was assigned as an unsigned variable instead of a signed variable. The output was still the same as If it was positive, but it came out as negative. This was done by the raspberry pi type casting the output to make it seem right and the code to not break.





The printing out of the cpsr register that houses the flags of the ARM Programming was kinda hard to figure out. The registers were always printed out in hexadecimal and trying to print

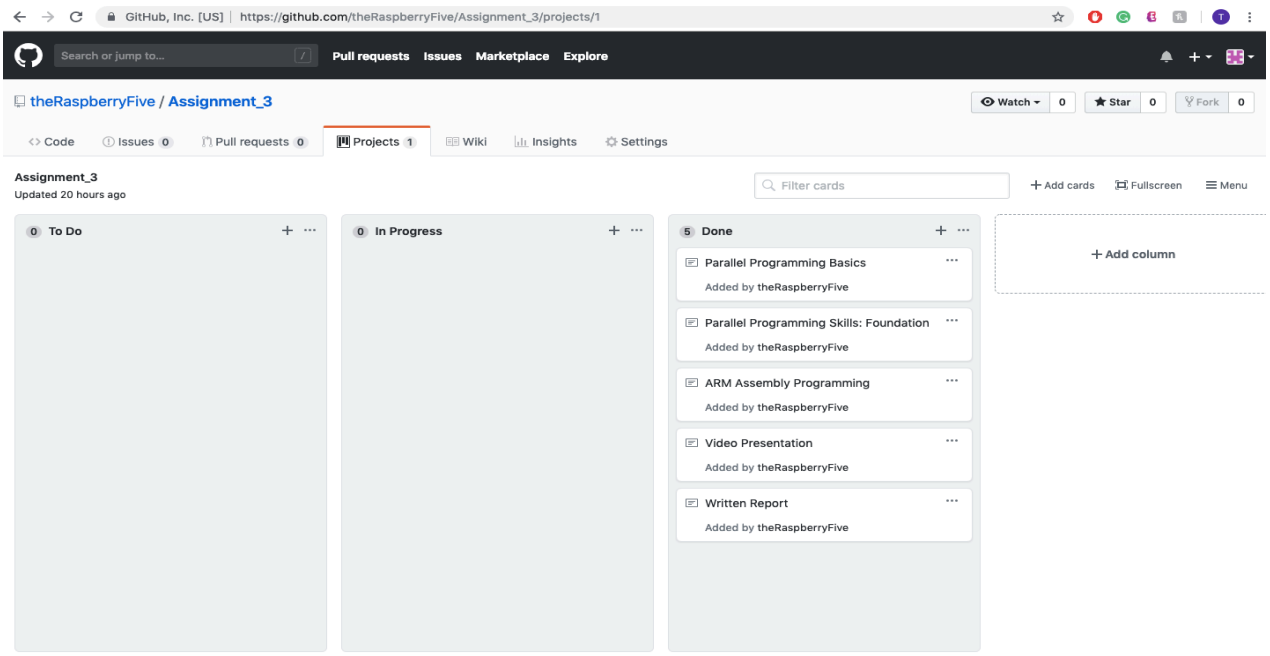
them out in binary to see the full 32 bits didn't exactly work out. They weren't taking up all 32 bits as they should have. I used another register to see the result and that one printed out all 32 bits.



Task 5

Appendix

- https://github.com/theRaspberryFive



- https://raspberryfive.slack.com/messages

Task 6

Presentation

- **https://youtu.be/QP93TfhJCG8**