Developing Soft and Parallel Programming Skills Using Project-Based Learning

Spring 2019

Group Name: ATLAS-SQUAD

Team Members: **Jason Poston (Team Coordinator)**
Sai Rampally
T'Avvion Jones
Zeak Sims
Shili Guan

Planning and Scheduling

Assignee Name	Email	Task	Duration (hours)	Dependency	Due Date	Note
Jason Poston (Team Lead)	jposton1@student.gsu.edu	Task 3 Foundation Task 3 Parallel Programing Basics Task 4 ARM Assembly Programing Task 5: Report/GitHub	4-6 hrs	None	02/19	All tasks must be finished and sent to lead by 2/19/19. Excellent (100%)
T'Avvion Jones	Tjones172@student.gsu.edu	Task 3a Q:3-5 Task 6 video Task 5: Submitting report via Google Docs/ GitHub	2 hrs	None	02/19	All tasks must be finished and sent to lead by 2/19/19. Excellent (100%)
Shili Guan	sguan2@student.gsu.edu	Task 3a Q:6,7 Task 6 video Task 5: Submitting report via Google Docs/ GitHub	2 hrs	None	02/19	All tasks must be finished and sent to lead by 2/19/19. Excellent (100%)
Zeak Sims	zsims2@student.gsu.edu	Task 3a Q:8,9 Task 5: Submitting report via Google Docs/ GitHub Task 6 video	2 hrs	None	02/19	All tasks must be finished and sent to lead by 2/19/19. Excellent (100%)
Sai Rampally	srampally1@student.gsu.edu	Task 3a Q 1+2 Task 6 (Upload YouTube Video) Assist with Google Docs/GitHub	2-3 hrs	YouTube channel Upload	02/19	All tasks must be finished and sent to lead by 2/19/19. Excellent (100%)

TeamWork Basics

1.) Identifying the components on the Raspberry PI B+

The Raspberry Pi B+ consists of a 1GB RAM, SD card slot, RCA jack, LED lights, CPU, 4 USB 2.0 ports, HDMI connection port, a 3.5 mm audio jack, Power cable slot, Camera jack, and a ethernet port/wireless internet LAN. The CPU is a 64-bit ARM multi-core, and the video output can also be connected to raspberry pi via HDMI. It supports a wireless LAN, and there is an ethernet jack which makes the pi to be able to connect through LAN wire.

2.) How many cores does the Raspberry Pi B+ CPU have

The raspberry pi has quad core processor (4 cores) that it can process multiple tasks.

3.) List three main differences between X86 (CISC) and ARM Raspberry PI (RISC). Justify your answer and use you own words (do not copy and paste).

The main and most significant difference between the two is the instruction set. The ARM has a smaller instruction set, it also contains more registers than the X86. When compared to the X86, ARM has more operations, and more addressing modes. ARM processor has a simple instruction set, and uses more registers when compared to the X86 CISC architecture. Because of its limited instruction set, it has to do many more operations than X86. This might seem like a disadvantage, however having a reduced instruction set, allows the ARM to execute more quickly, though harder (longer) to write. In ARM, instruction can be used for conditional execution.

4.) What is the difference between sequential and parallel computation and identify the practical significance of each?

The difference between sequential and parallel computation is that in serial computing a problem is broken up into a series of instructions and executed one after the other in a sequential order on a single processor. Whereas in parallel programming, a problem is broken into small pieces that are then broken into smaller instructions that will be solved simultaneously yet executed on different processors.

5.) Identify the basic form of data and task parallelism in computational problems.

The basic form of data and task parallelism in computation problems is:
Data Parallelism is the simultaneous execution on multiple cores of the same functions across the elements of the dataset. While in task parallelism, it also simultaneous executed on multiple cores of many functions across the same of different datasets.

6.) <u>Explain the difference between processes and threads.</u>

A process is the abstraction of a running program that don't share memory with each other; Thread is a lightweight process that allows a single executable/process to be decomposed to smaller and independent parts. Thread shares the common memory of the process they belong to. Therefore, the difference between processes and threads is that while processes don't share memory, the threads share common memory of the process which they belong to.

7.) <u>What is OpenMP and what is OpenMP pragmas?</u>

OpenMP (Open Multi-Processing) is an application programming interface, it uses an implicit multithreading model in which the library handles thread creation and management. OpenMP can reduce the complex and errors of the task. OpenMP pragmas is compiler directives that enable the compiler to generate threaded code. OpenMP programs also use multiple instruction, multiple data pattern additional to the Thread pool pattern.

8.) <u>What applications benefit from multi-core (list four)</u>

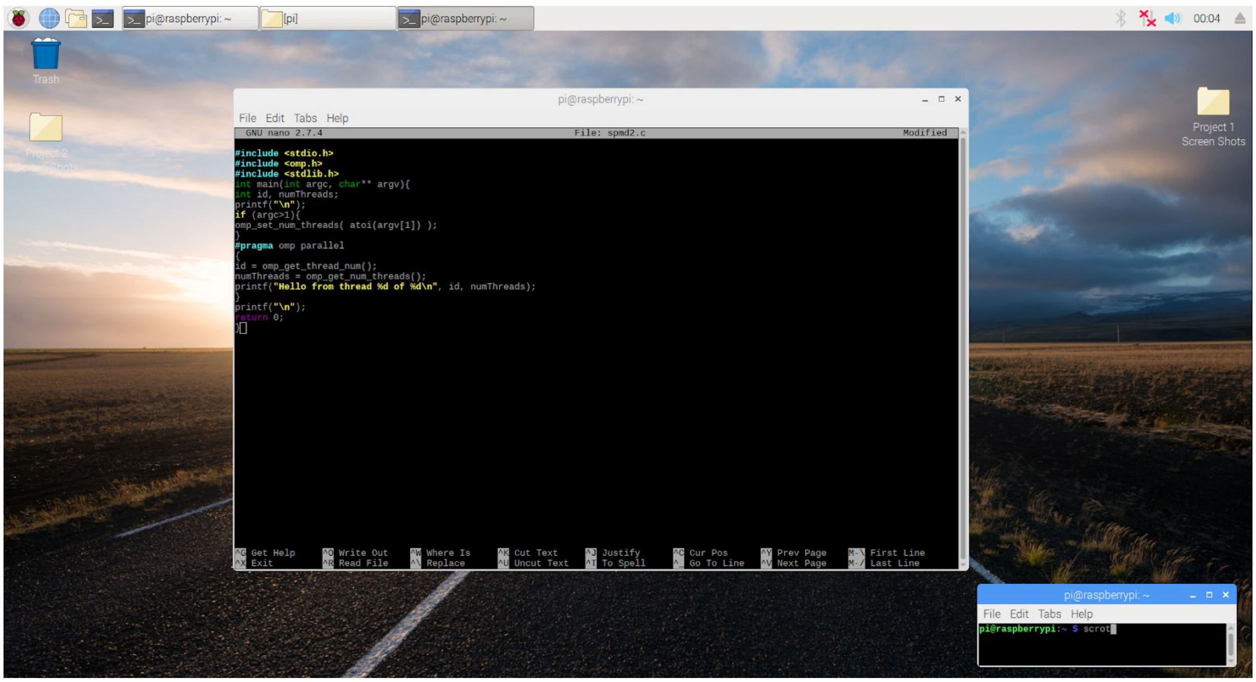
The applications that benefit from multi-core are Database servers, compilers, multimedia applications, and web services/commerce.

9.) <u>Why Multicore?</u>

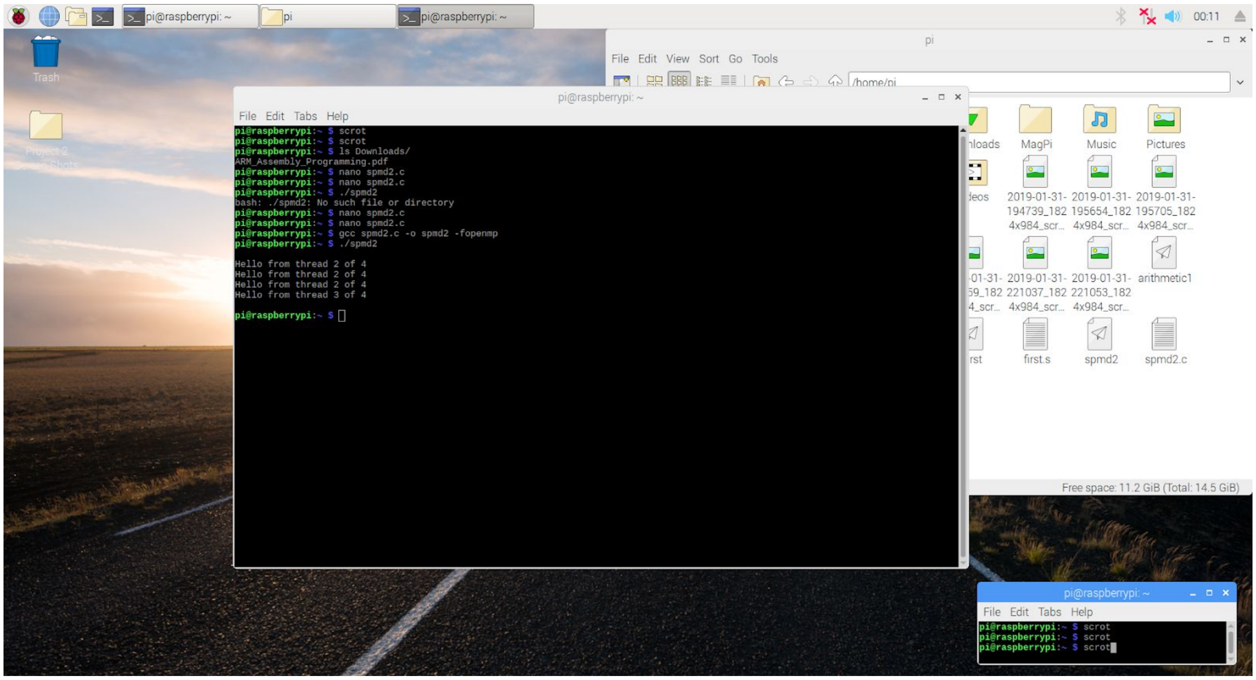
Multi-core architectures have many advantages over the single core architecture. Most major OS support multi-core and perceive each core as a separate processor. This means the OS can execute more processes at one time also known as concurrency. There are a couple of disadvantages with the single-core architecture. For one, it is difficult to make single-core clock frequencies even higher. Many new applications also are multithread, and there has be more of a shift towards more parallelism. Another disadvantage of the single-core architecture is that it leads to overheating.

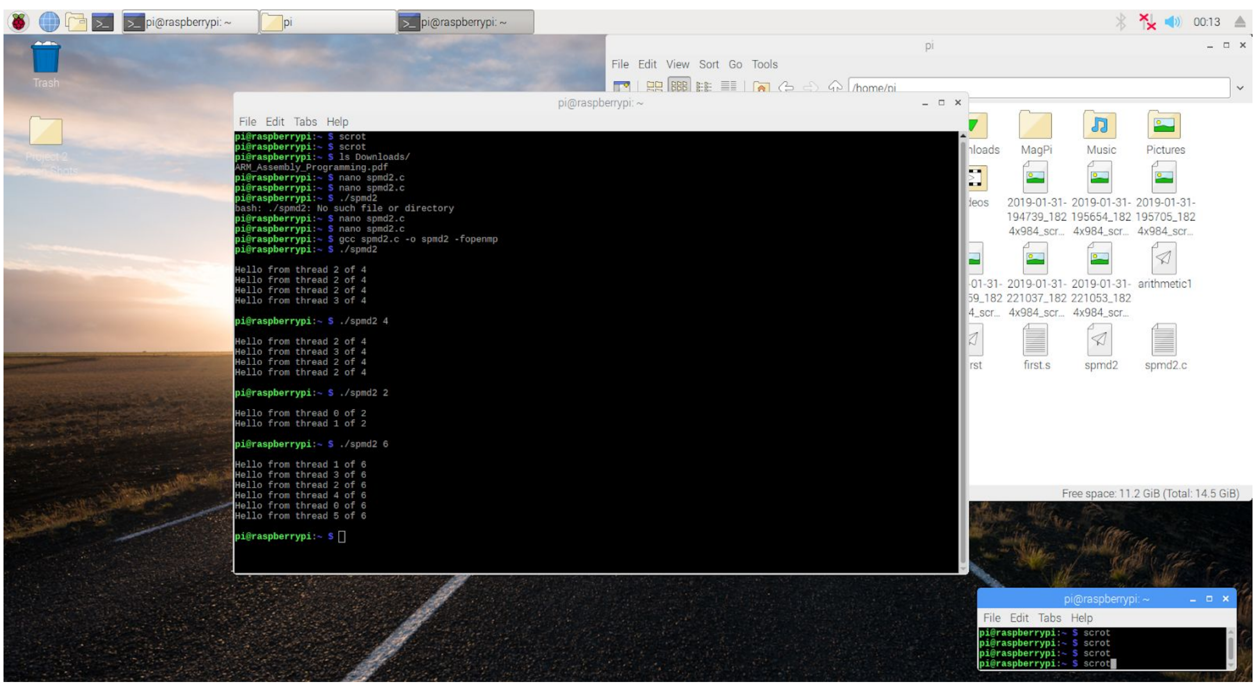
Parallel Programming Basics

The initial tasks were to become more familiar with the Raspberry Pi system, and the Terminal that we will use to code stuff in. After following along with a few instructions on how to maneuver better inside the Terminal, I began coding using the spmd2 template.

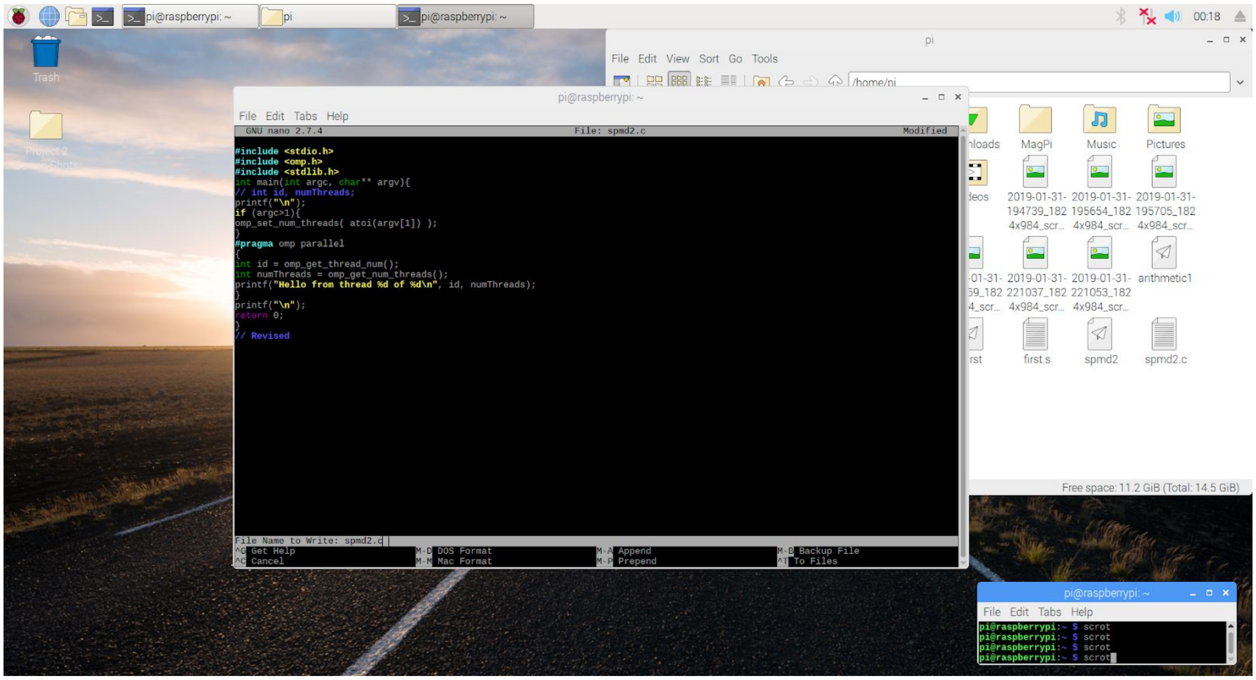


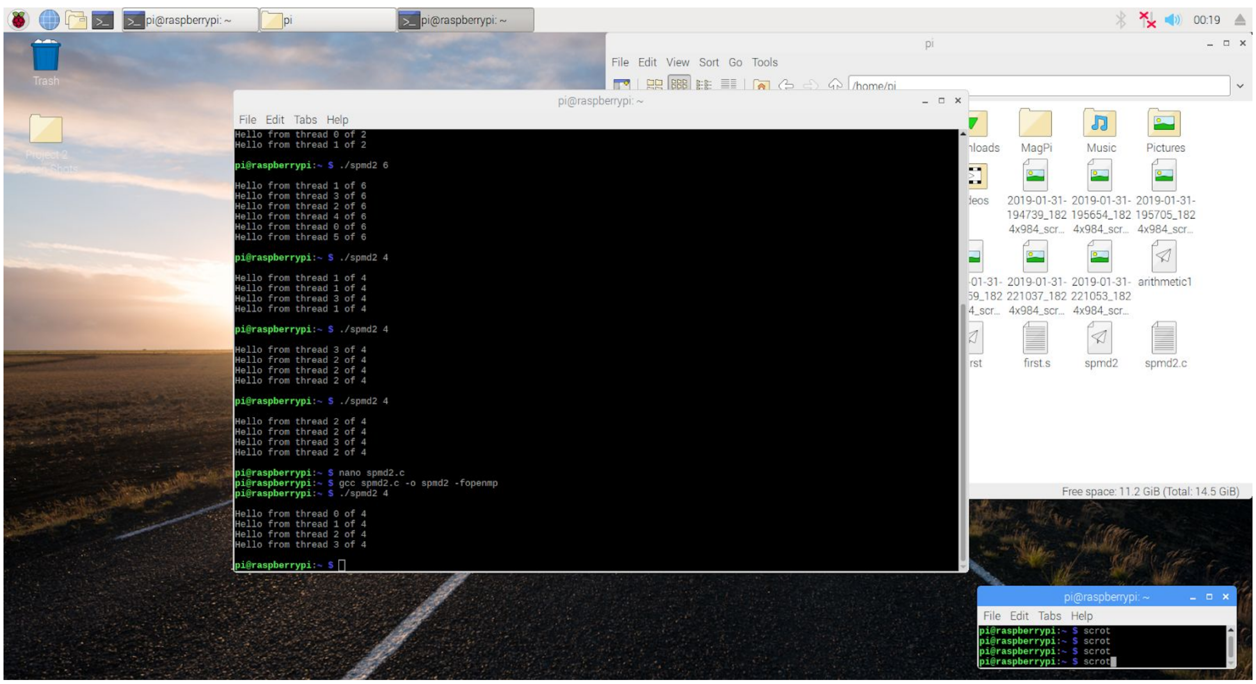
Line 10 (#pragma comp parallel) here was the focus, as it is what initiate the 'fork' thread. This allows line 11-15 to perform different tasks concurrently. Anything inside the "{}" would be run on different forks and then brought back together.





Calling ./spmd2 4 (along with various other numbers) allowed me to see exactly what core was finishing it's task first, which varied from iteration to iteration. The issue being that sometimes the same core was being used. This is due to the cores sharing a bank of memory in the machine. I then went back in and fixed the code in order to better this process, making sure each thread was being handled by one and not multiple cores.

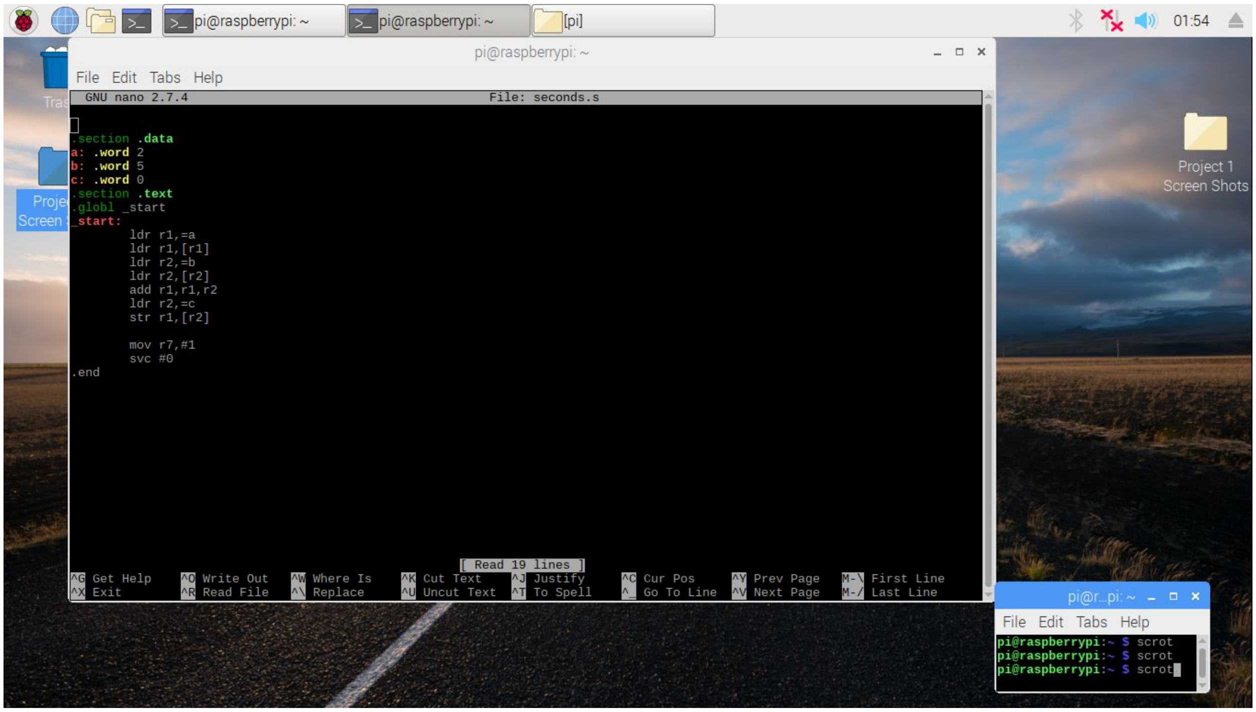




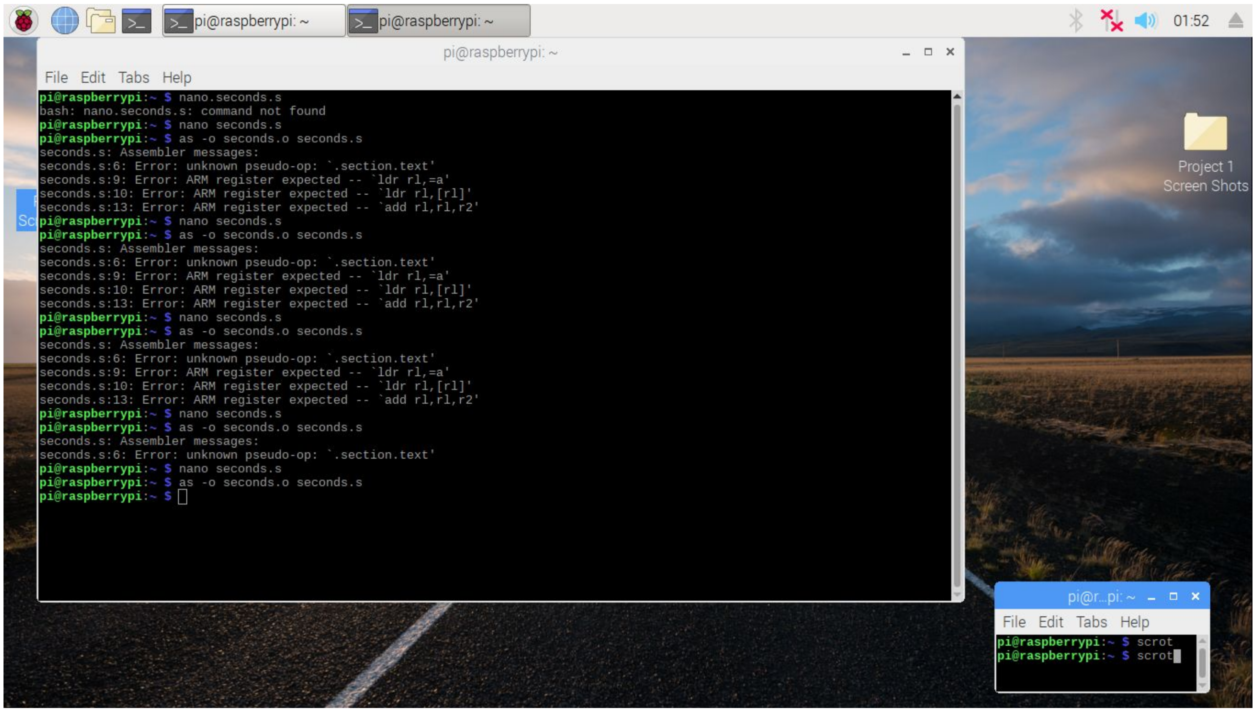
Now, while they're not in sequential order, only one core was used per thread, no matter how many I typed in (tried a few just to see).

ARM Assembly Programming

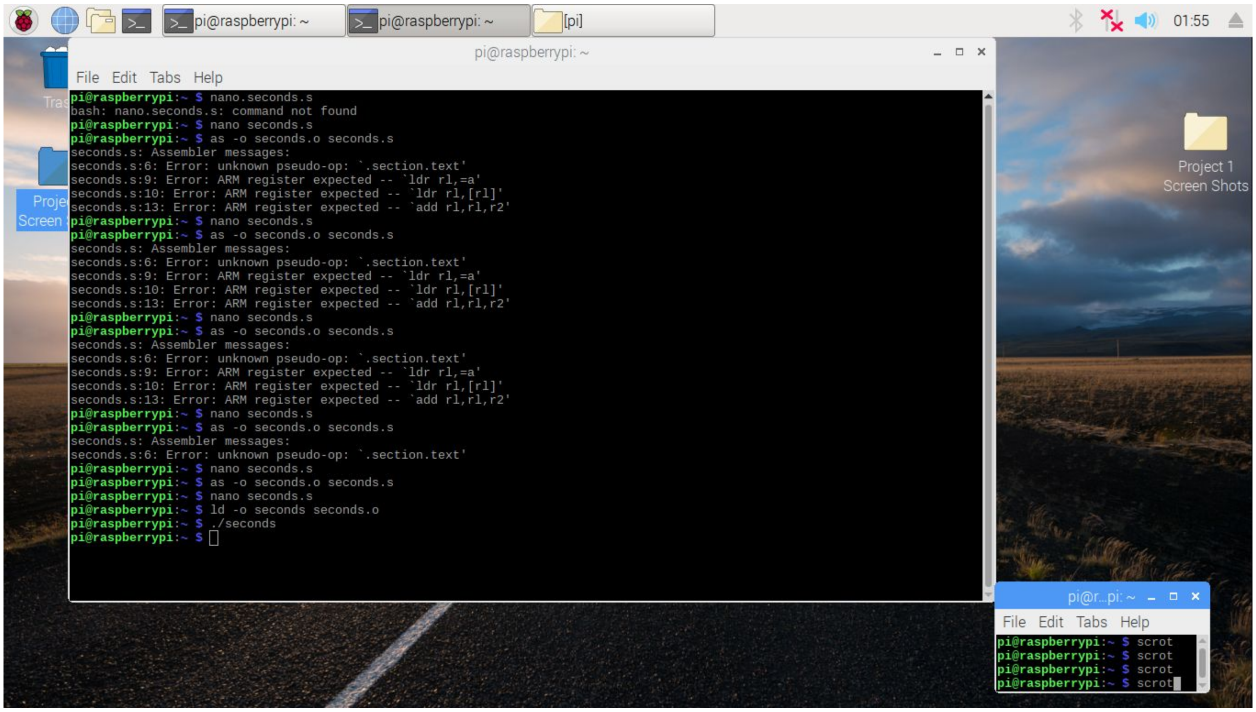
The second task was to copy the code from the handout and link it.

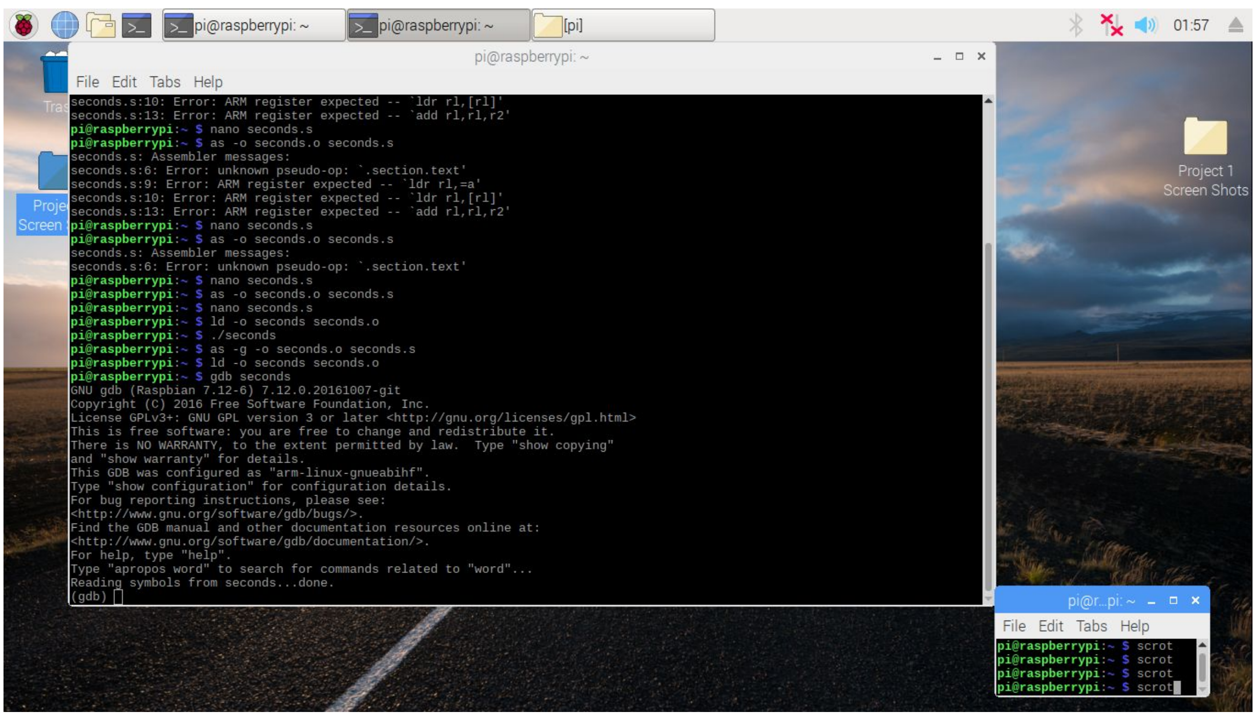


Once linked, the program ran, however didn't produce any output. We needed run the debugger to see what was going on with the actual program.



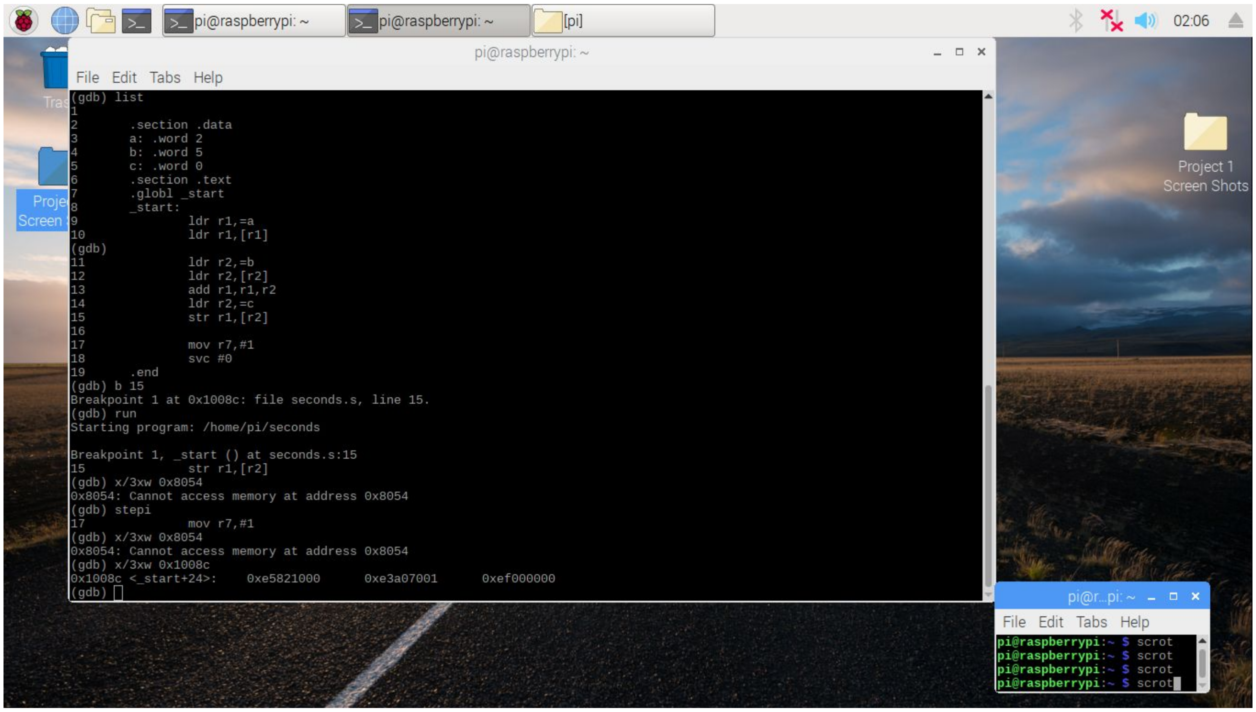
After adding the flags to the assembler line command, linking it again, and running the debugger we were able to apply breakpoints to have a better understanding of what was being stored and where.



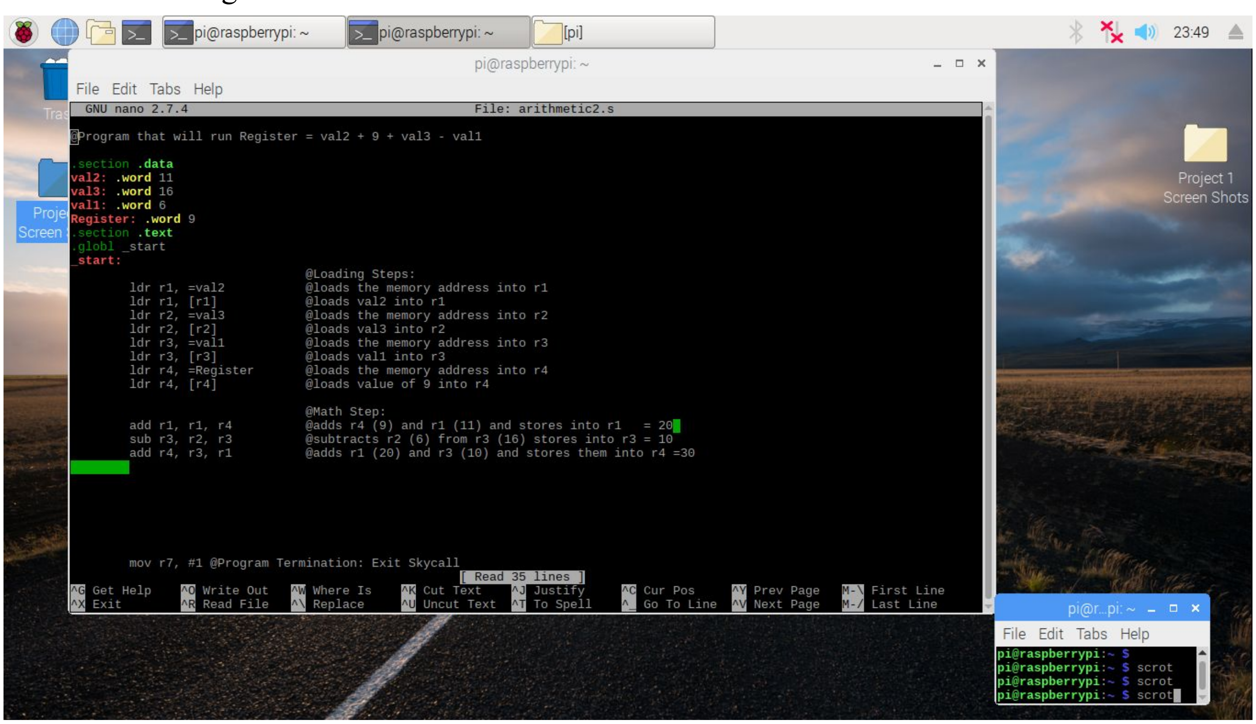


The breakpoint was applied to Line 15 (the Storing step). Here we learned how to read and ask for certain information from the register. You can ask for the format and size in various way (like hexbyte or bitword). In this exercise we were curious about the Hex of our register which was 0x1008c, displayed below. This allowed us to see what was stored in there after the breakpoint which allows us to better follow and see where the values are in our memory.

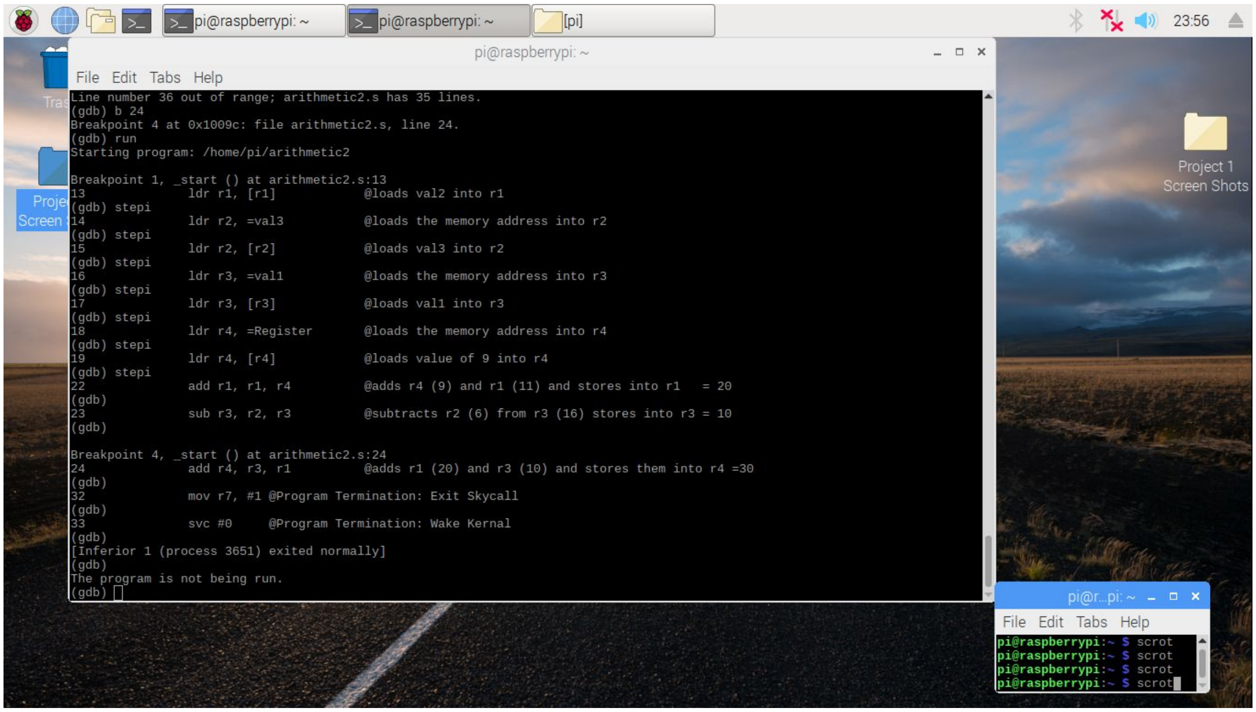
0x35821000



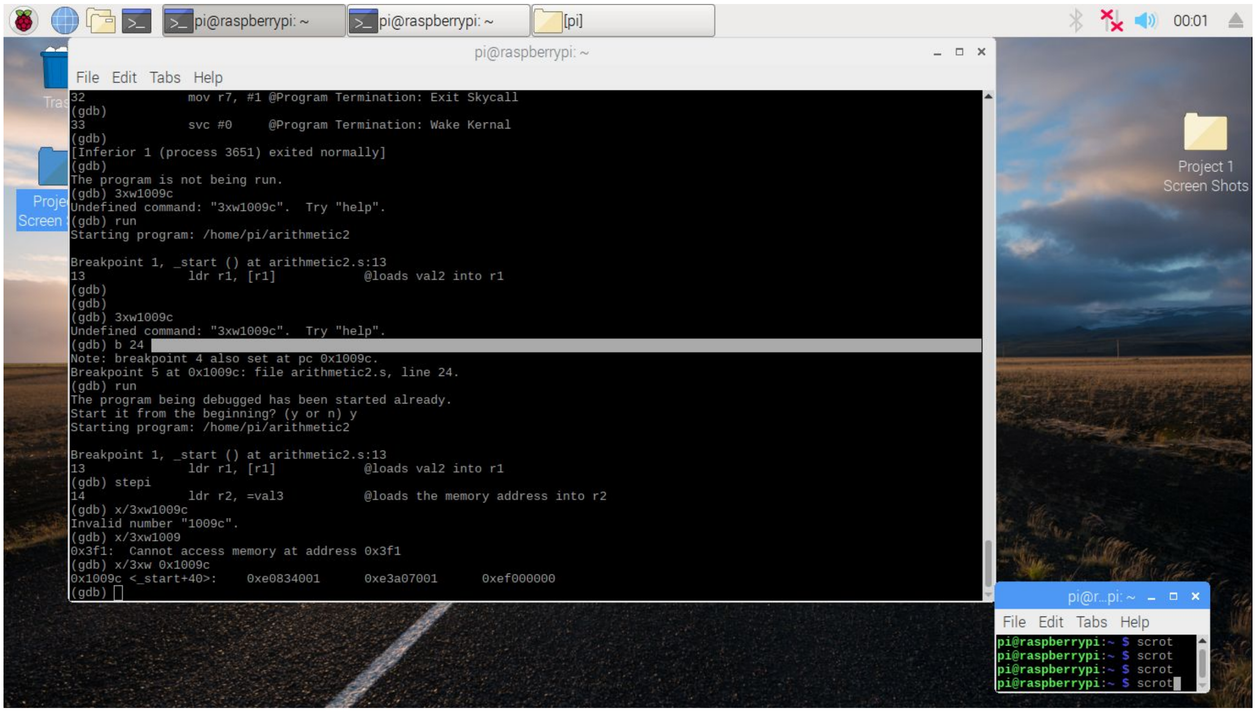
The second part of the assignment was to write our own code that would add values together and store them in a register.



Again, we were to write, assemble, link and debug the code



Then calling back what we just learned, obtain the Hex code from the memory address of 0x1009c, which was 0xe0834001. Again, we were able to observe the value located at the specific address using the debugger. Looking back at the pervious project, we also checked using info register prompt to bring up a list and their stored values. This ensured that what expected to be there was in fact there.


pi@raspberrypi: ~
pi@raspberrypi: ~
[pi]
00:01
pi@raspberrypi: ~
File Edit Tabs Help
32 mov r7, #1 @Program Termination: Exit Skycall
(gdb)
33 svc #0 @Program Termination: Wake Kernal
(gdb)
[Inferior 1 (process 3651) exited normally]
(gdb)
The program is not being run.
(gdb) 3xw1009c
Undefined command: "3xw1009c". Try "help".
(gdb) run
Starting program: /home/pi/arithmetic2
Breakpoint 1, _start () at arithmetic2.s:13
13 ldr r1, [r1] @loads val2 into r1
(gdb)
(gdb)
(gdb) 3xw1009c
Undefined command: "3xw1009c". Try "help".
(gdb) b 24
Note: breakpoint 4 also set at pc 0x1009c.
Breakpoint 5 at 0x1009c: file arithmetic2.s, line 24.
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/pi/arithmetic2
Breakpoint 1, _start () at arithmetic2.s:13
13 ldr r1, [r1] @loads val2 into r1
(gdb) stepi
14 ldr r2, =val3 @loads the memory address into r2
(gdb) x/3xw1009c
Invalid number "1009c".
(gdb) x/3xw1009
0x3f1: Cannot access memory at address 0x3f1
(gdb) x/3xw 0x1009c
0x1009c <_start+40>: 0xe0834001 0xe3a07001 0xef000000
(gdb)
Project 1
Screen Shots
pi@r...pi: ~
File Edit Tabs Help
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot

Appendix

Slack Account: https://atlas-squad.slack.com/messages/CFR6D9LHJ/

GitHub: https://github.com/ATLAS-SQUAD/CSc-3210

YouTube link: https://www.youtube.com/watch?v=L50Ig93PXsY