

Developing Soft and Parallel Programming Skills Using Project- Based Learning

TheRaspberryFive

Jay Patel, Marcus Aqui, Bonnie Atelsek, Jacques 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
Jacques 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

The screenshot shows a GitHub project board for the repository 'theRaspberryFive / Assignment_3'. The board has three columns: 'To Do', 'In Progress', and 'Done'. The 'Done' column contains five cards:

- Parallel Programming Basics
- Parallel Programming Skills: Foundation
- ARM Assembly Programming
- Video Presentation
- Written Report

Each card includes a link icon and the text 'Added by theRaspberryFive'.

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.

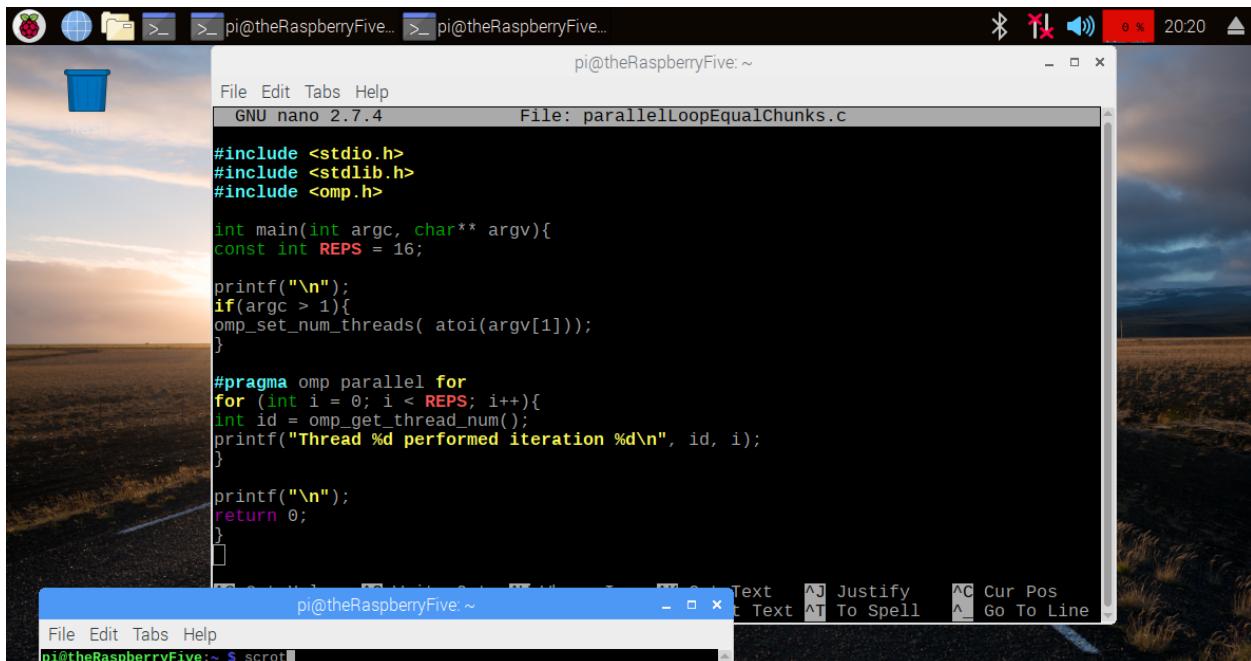
What is system on chip (SoC)? Does Raspberry PI use system on SoC?

- 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.

```

pi@theRaspberryFive:~ $ nano parallelLoopEqualChunks.c
pi@theRaspberryFive:~ $ gcc parallelLoopEqualChunks.c -o pLoop -fopenmp
pi@theRaspberryFive:~ $ nano parallelLoopEqualChunks.c
pi@theRaspberryFive:~ $ ./pLoop 4
Thread 2 performed iteration 8
Thread 2 performed iteration 9
Thread 2 performed iteration 10
Thread 2 performed iteration 11
Thread 1 performed iteration 4
Thread 1 performed iteration 5
Thread 3 performed iteration 12
Thread 3 performed iteration 13
Thread 3 performed iteration 14
Thread 3 performed iteration 15
Thread 1 performed iteration 6
Thread 1 performed iteration 7
Thread 0 performed iteration 0
Thread 0 performed iteration 1
Thread 0 performed iteration 2
Thread 0 performed iteration 3
pi@theRaspberryFive:~ $ scrot
pi@theRaspberryFive:~ $ scrot

```

```

pi@theRaspberryFive:~ $ nano parallelLoopEqualChunks.c
pi@theRaspberryFive:~ $ gcc parallelLoopEqualChunks.c -o pLoop -fopenmp
pi@theRaspberryFive:~ $ ./pLoop 4
Thread 0 performed iteration 1
Thread 0 performed iteration 2
Thread 0 performed iteration 3
Thread 2 performed iteration 9
Thread 2 performed iteration 10
Thread 0 performed iteration 0
Thread 3 performed iteration 13
Thread 3 performed iteration 14
Thread 3 performed iteration 15
Thread 2 performed iteration 11
Thread 2 performed iteration 12
Thread 1 performed iteration 5
Thread 1 performed iteration 6
Thread 3 performed iteration 16
Thread 0 performed iteration 1
Thread 0 performed iteration 2
Thread 0 performed iteration 3
Thread 0 performed iteration 4
Thread 1 performed iteration 7
Thread 1 performed iteration 8
pi@theRaspberryFive:~ $ scrot
pi@theRaspberryFive:~ $ scrot

```

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.

```

pi@theRaspberryFive: ~
pi@theRaspberryFive: ~$ scrot

```

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

```

pi@theRaspberryFive: ~
pi@theRaspberryFive: ~$ gcc parallelLoopChunksOf1.c -o pLoop2 -fopenmap
pi@theRaspberryFive: ~$ gcc parallelLoopChunksOf1.c -o pLoop2 -fopenmp
pi@theRaspberryFive: ~$ nano parallelLoopChunksOf1.c
pi@theRaspberryFive: ~$ ./pLoop2 4
bash: ./pLoop2: No such file or directory
pi@theRaspberryFive: ~$ ./pLoop2 4

Thread 1 performed iteration 1
Thread 1 performed iteration 5
Thread 1 performed iteration 9
Thread 1 performed iteration 13
Thread 2 performed iteration 2
Thread 2 performed iteration 6
Thread 2 performed iteration 10
Thread 2 performed iteration 14
Thread 0 performed iteration 0
Thread 0 performed iteration 4
Thread 0 performed iteration 8
Thread 0 performed iteration 12
Thread 3 performed iteration 3
Thread 3 performed iteration 7
Thread 3 performed iteration 11
Thread 3 performed iteration 15

pi@theRaspberryFive: ~$ 

```

```

pi@theRaspberryFive: ~
pi@theRaspberryFive: ~$ scrot

```

```

pi@theRaspberryFive:~ $ gcc parallelLoopChunksOf1.c -o ploop2 -fopenmap
gcc: error: unrecognized command line option '-fopenmap'; did you mean '-fopenmp'?
pi@theRaspberryFive:~ $ gcc parallelLoopChunksOf1.c -o ploop2 -fopenmp
gcc: error: unrecognized command line option '-fopenmap'; did you mean '-fopenmp'?
pi@theRaspberryFive:~ $ nano parallelLoopChunksOf1.c
pi@theRaspberryFive:~ $ ./ploop2 4
bash: ./ploop2: No such file or directory
pi@theRaspberryFive:~ $ ./ploop2 4

Thread 1 performed iteration 1
Thread 1 performed iteration 5
Thread 1 performed iteration 9
Thread 1 performed iteration 13
Thread 2 performed iteration 2
Thread 2 performed iteration 6
Thread 2 performed iteration 10
Thread 2 performed iteration 14
Thread 0 performed iteration 0
Thread 0 performed iteration 4
Thread 0 performed iteration 8
Thread 0 performed iteration 12
Thread 3 performed iteration 3
Thread 3 performed iteration 7
Thread 3 performed iteration 11
Thread 3 performed iteration 15

pi@theRaspberryFive:~ $ scrot

```

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.

```

pi@theRaspberryFive:~ $ nano reduction.c
pi@theRaspberryFive:~ $ gcc reduction.c -o reduction -fopenmp
reduction.c: In function 'main':
reduction.c:18:1: warning: implicit declaration of function 'intialize' [-Wimplicit-function-declaration]
    intialize(array, SIZE);
    ^~~~~~
/tmp/ccdXXC6w.o: In function `main':
reduction.c:(.text+0x7c): undefined reference to `intialize'
collect2: error: ld returned 1 exit status
pi@theRaspberryFive:~ $ ./reduction 4
bash: ./reduction: No such file or directory
pi@theRaspberryFive:~ $ nano reduction.c
pi@theRaspberryFive:~ $ gcc reduction.c -o reduction -fopenmp
reduction.c: In function 'main':
reduction.c:18:1: warning: implicit declaration of function 'intialize' [-Wimplicit-function-declaration]
    intialize(array, SIZE);
    ^~~~~~
/tmp/cceFdPUH.o: In function `main':
reduction.c:(.text+0x7c): undefined reference to `intialize'
collect2: error: ld returned 1 exit status
pi@theRaspberryFive:~ $ nano reduction.c

```

```

pi@theRaspberryFive:~ $ scrot

```

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 screenshot shows a terminal window on a Raspberry Pi. The title bar says "pi@theRaspberryFive... > pi@theRaspberryFive...". The terminal window contains the following code:

```

File Edit Tabs Help
GNU nano 2.7.4          File: reduction.c          Modified
#include <stdio.h>
#include <omp.h>
#include <stdlib.h>

void initialize(int* a, int n);
int sequentialSum(int* a, int n);
int parallelSum(int* a, int n);

#define SIZE 1000000

int main(int argc, char** argv){
int array[SIZE];

if (argc > 1){
omp_set_num_threads(atoi(argv[1]));
}

initialize(array, SIZE);
printf("\nSequential sum: %d\nParallel sum: %d\n\n", sequentialSum(array, SIZE));
return 0;
}

/*fill array with random values*/

```

Below the code, there is a status bar with the following options: ^G Get Help, ^O Write Out, ^W Where Is, ^X Exit, ^R Read File, ^N Replace. To the right of the terminal window, the command line shows the output of the program:

```

pi@theRaspberryFive:~ $ scrot

```

The screenshot shows a terminal window on a Raspberry Pi. The title bar says "pi@theRaspberryFive... > pi@theRaspberryFive...". The terminal window contains the following modified code:

```

File Edit Tabs Help
GNU nano 2.7.4          File: reduction.c          Modified
for(i = 0; i < n; i++){
a[i] = rand() % 1000;
}

/*sum the array sequentially*/
int sequentialSum(int* a, int n){
int sum = 0;
int i;
for(i = 0; i < n; i++){
sum += a[i];
}
return sum;
}

/*sum the array using multiple threads*/
int parallelSum(int* a, int n){
int sum = 0;
int i;
//pragma omp parallel for //reduction(+=sum)
for (i=0;i<n;i++){
sum+= a[i];
}

```

Below the code, there is a status bar with the following options: ^G Get Help, ^O Write Out, ^W Where Is, ^X Exit, ^R Read File, ^N Replace. To the right of the terminal window, the command line shows the output of the program:

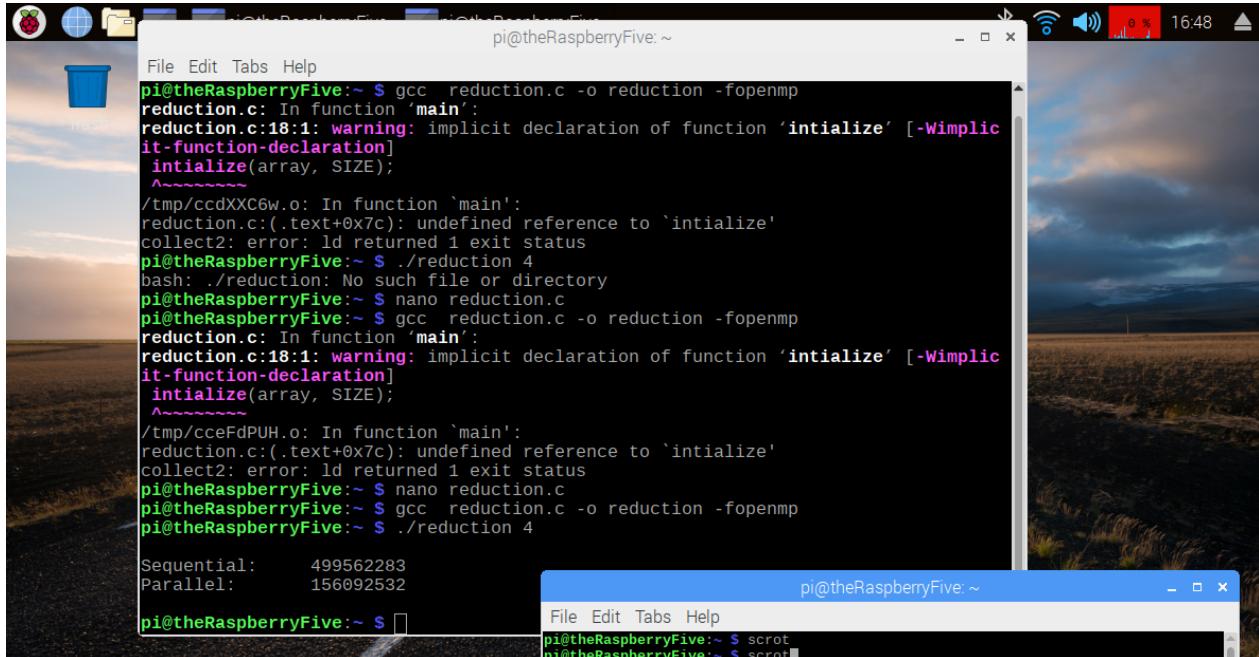
```

pi@theRaspberryFive:~ $ scrot
pi@theRaspberryFive:~ $ scrot

```

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.



```

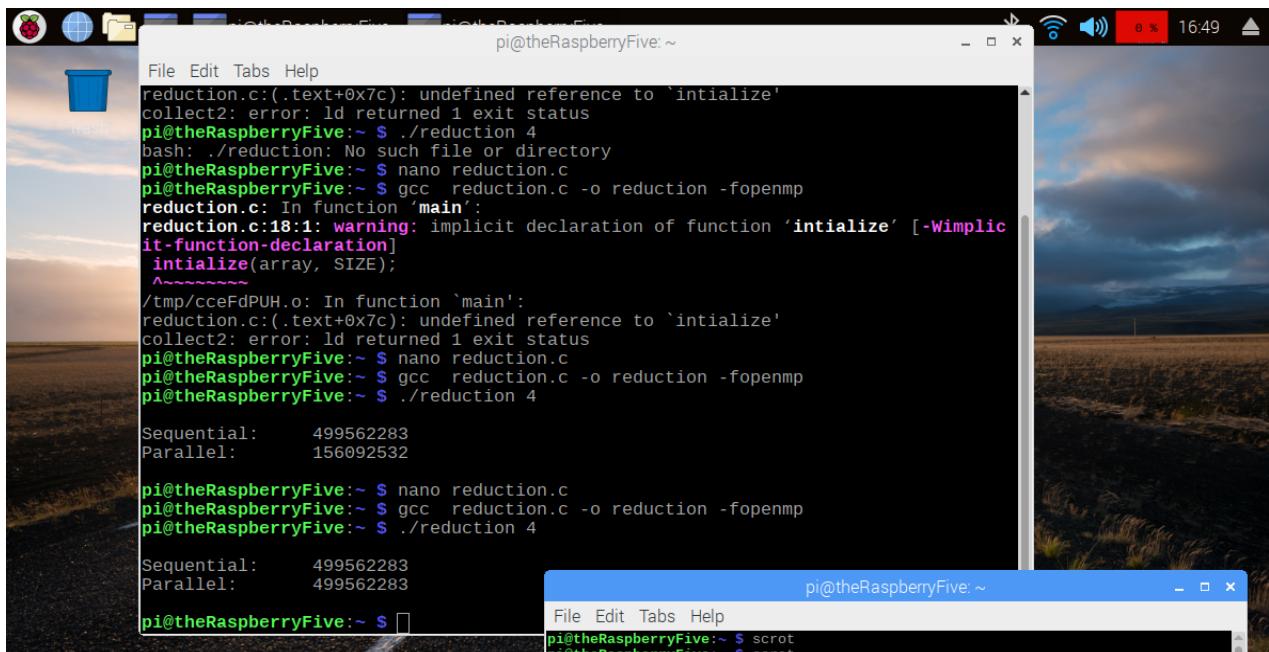
pi@theRaspberryFive:~ $ gcc reduction.c -o reduction -fopenmp
reduction.c: In function 'main':
reduction.c:(.text+0x7c): undefined reference to 'intialize' [-Wimplicit-function-declaration]
  intialize(array, SIZE);
^~~~~~

/tmp/ccdXXC6W.o: In function `main':
reduction.c:(.text+0x7c): undefined reference to `intialize'
collect2: error: ld returned 1 exit status
pi@theRaspberryFive:~ $ ./reduction 4
bash: ./reduction: No such file or directory
pi@theRaspberryFive:~ $ nano reduction.c
pi@theRaspberryFive:~ $ gcc reduction.c -o reduction -fopenmp
reduction.c: In function 'main':
reduction.c:18:1: warning: implicit declaration of function 'intialize' [-Wimplicit-function-declaration]
  intialize(array, SIZE);
^~~~~~

/tmp/cceFdPUH.o: In function `main':
reduction.c:(.text+0x7c): undefined reference to `intialize'
collect2: error: ld returned 1 exit status
pi@theRaspberryFive:~ $ nano reduction.c
pi@theRaspberryFive:~ $ gcc reduction.c -o reduction -fopenmp
pi@theRaspberryFive:~ $ ./reduction 4

Sequential:      499562283
Parallel:       156092532
  
```

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



```

reduction.c:(.text+0x7c): undefined reference to `intialize'
collect2: error: ld returned 1 exit status
pi@theRaspberryFive:~ $ ./reduction 4
bash: ./reduction: No such file or directory
pi@theRaspberryFive:~ $ nano reduction.c
pi@theRaspberryFive:~ $ gcc reduction.c -o reduction -fopenmp
reduction.c: In function `main':
reduction.c:18:1: warning: implicit declaration of function 'intialize' [-Wimplicit-function-declaration]
  intialize(array, SIZE);
^~~~~~

/tmp/cceFdPUH.o: In function `main':
reduction.c:(.text+0x7c): undefined reference to `intialize'
collect2: error: ld returned 1 exit status
pi@theRaspberryFive:~ $ nano reduction.c
pi@theRaspberryFive:~ $ gcc reduction.c -o reduction -fopenmp
pi@theRaspberryFive:~ $ ./reduction 4

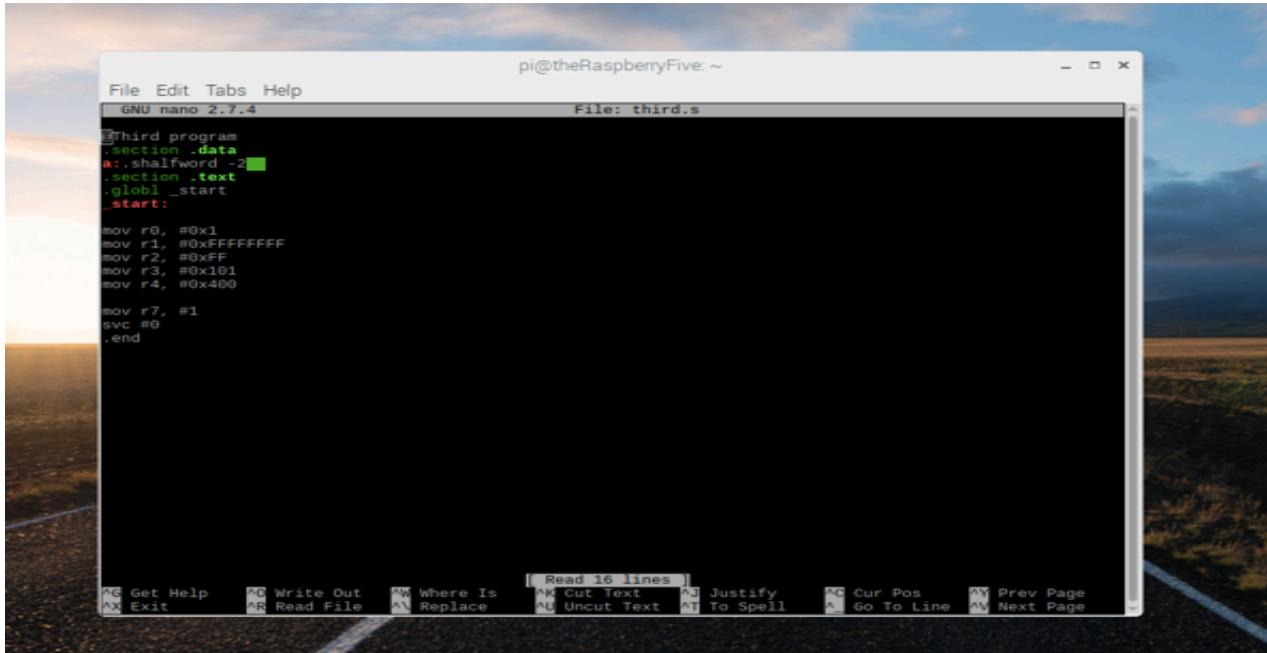
Sequential:      499562283
Parallel:       499562283
  
```

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.



pi@theRaspberryFive: ~

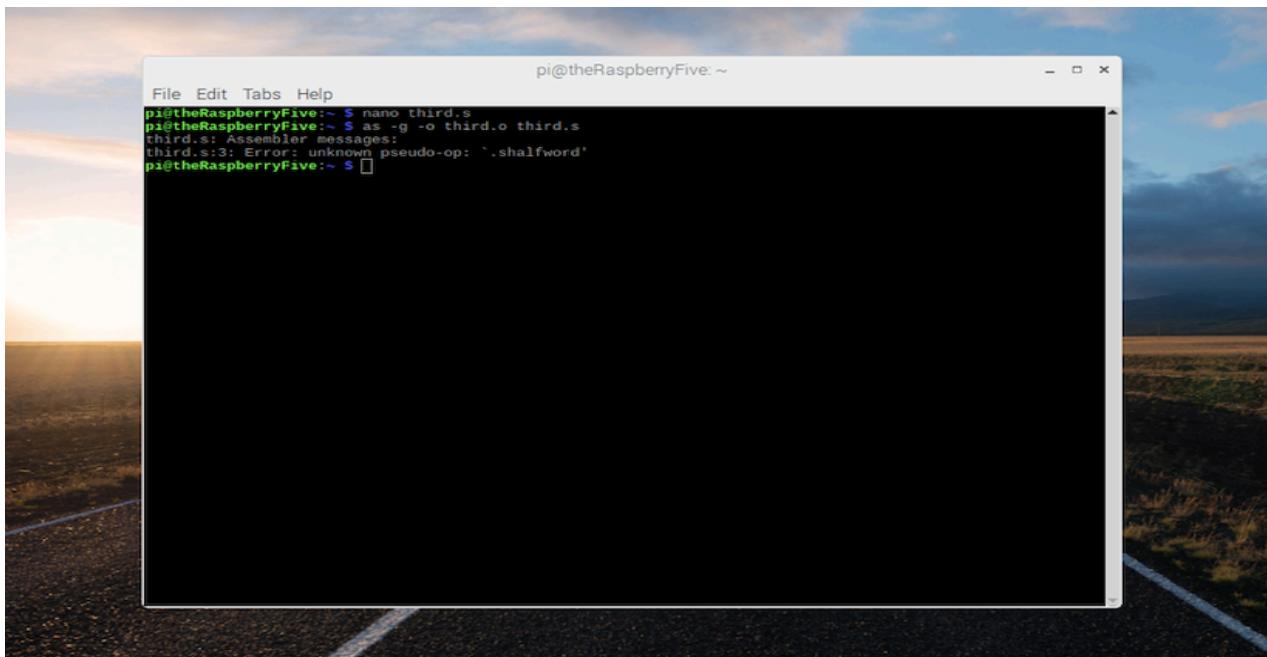
GNU nano 2.7.4 File: third.s

```
third program
.data
a:.halfword -2
.text
.global _start
_start:
    mov r0, #0x1
    mov r1, #0xFFFFFFFF
    mov r2, #0xFF
    mov r3, #0x101
    mov r4, #0x400
    mov r7, #1
    svc #0
.end
```

File Edit Tabs Help

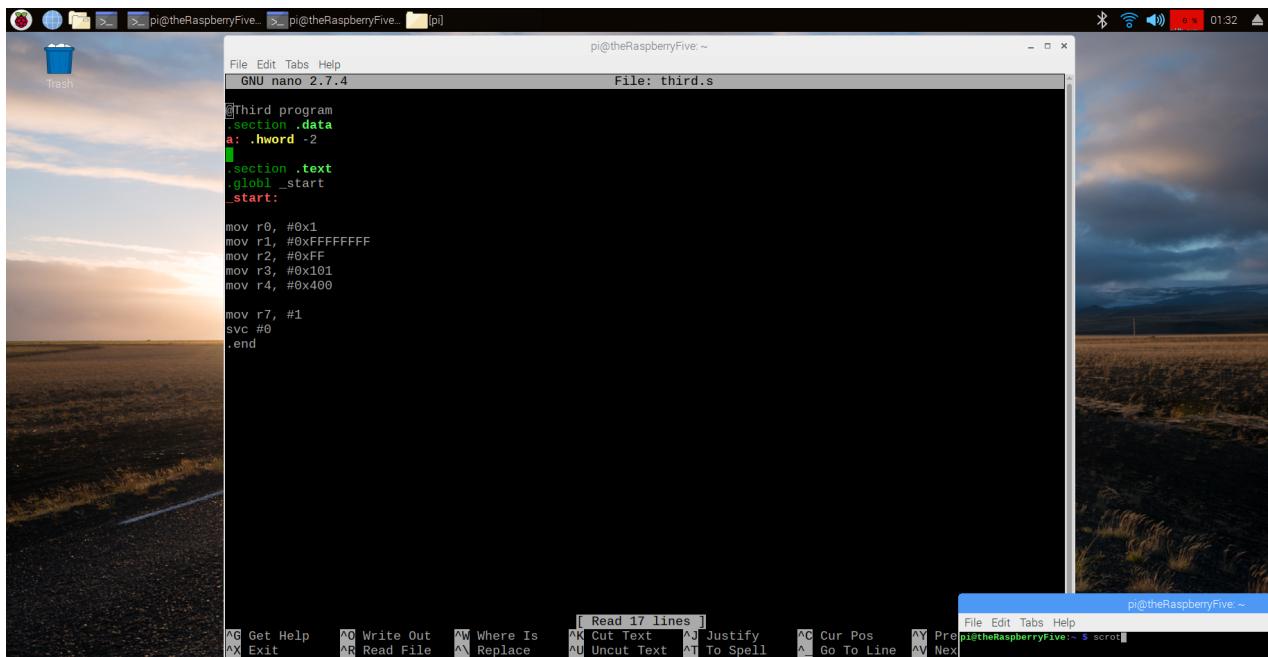
Get Help Write Out Where Is Read 16 lines Cut Text Justify Cur Pos Prev Page

Exit Read File Replace Uncut Text To Spell Go To Line Next Page

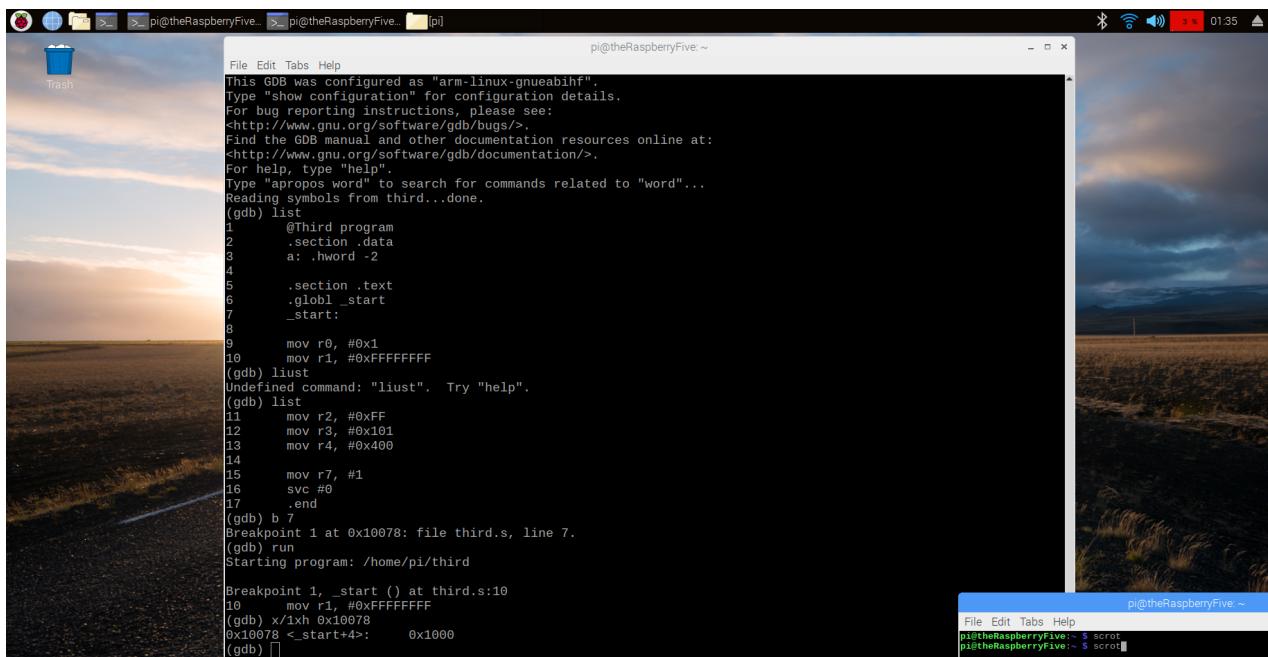


pi@theRaspberryFive: ~

```
pi@theRaspberryFive: ~$ nano third.s
pi@theRaspberryFive: ~$ as -g -o third.o third.s
third.s: Assembler messages:
third.s:3: Error: unknown pseudo-op: `;.halfword'
pi@theRaspberryFive: ~$
```

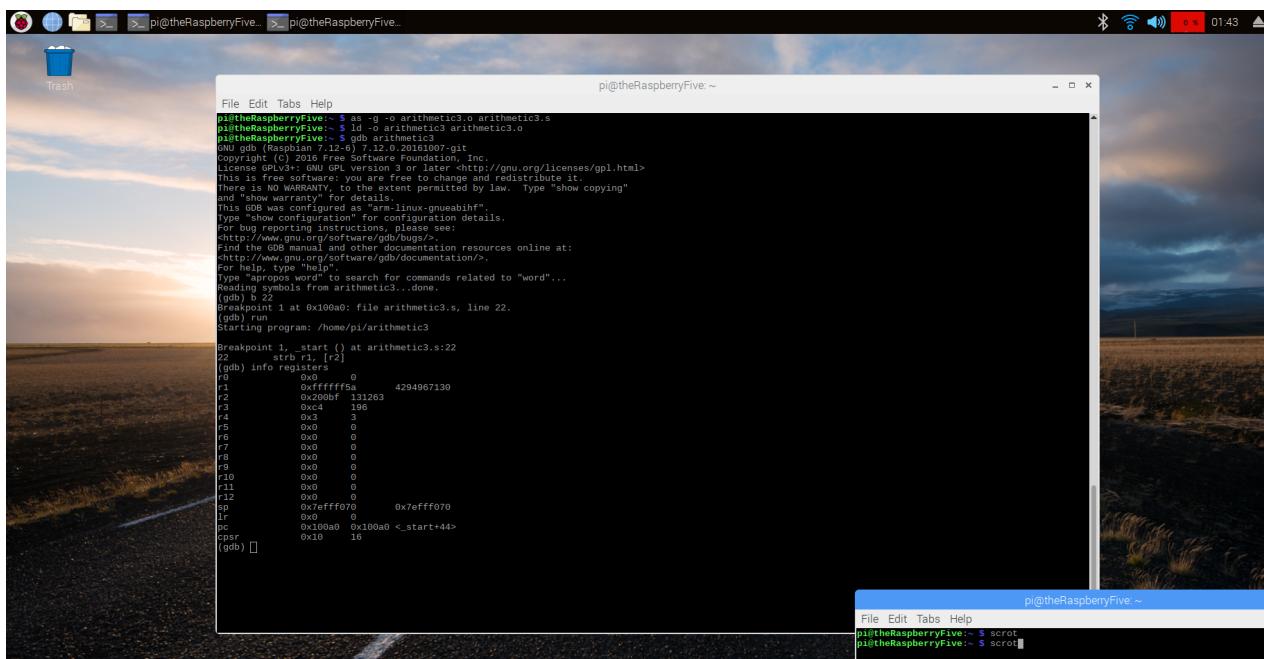
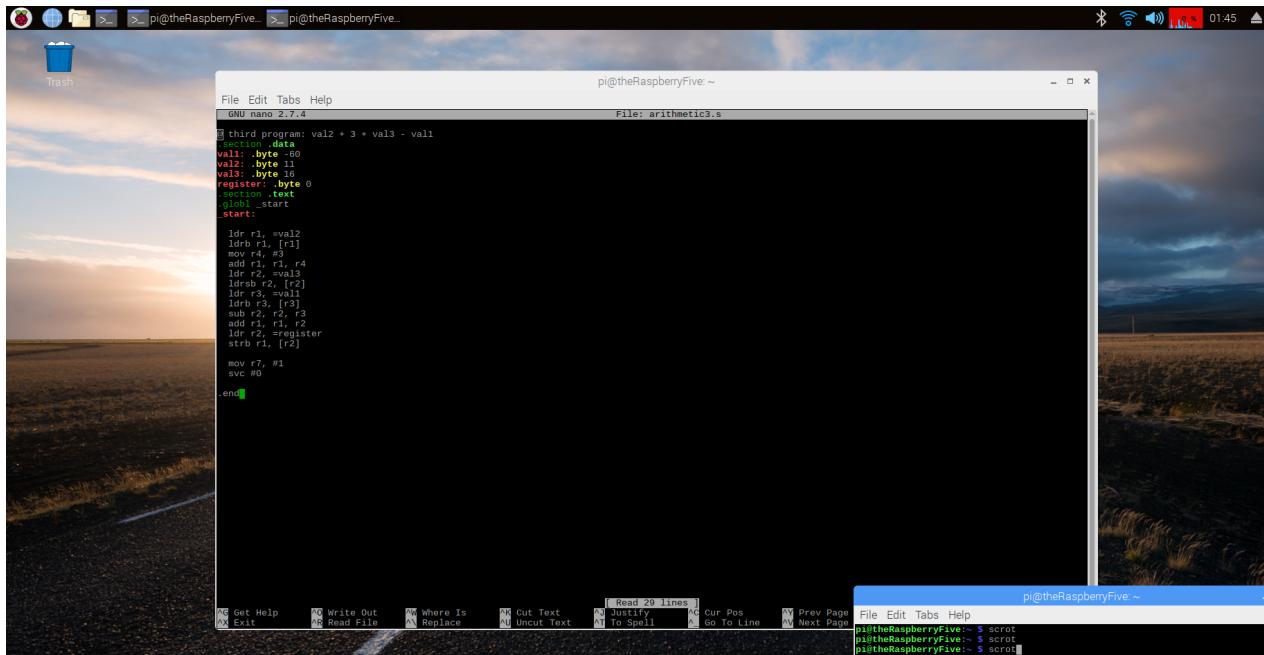


Printing out the hex value at a certain point in the was not too hard in this part of the programming.



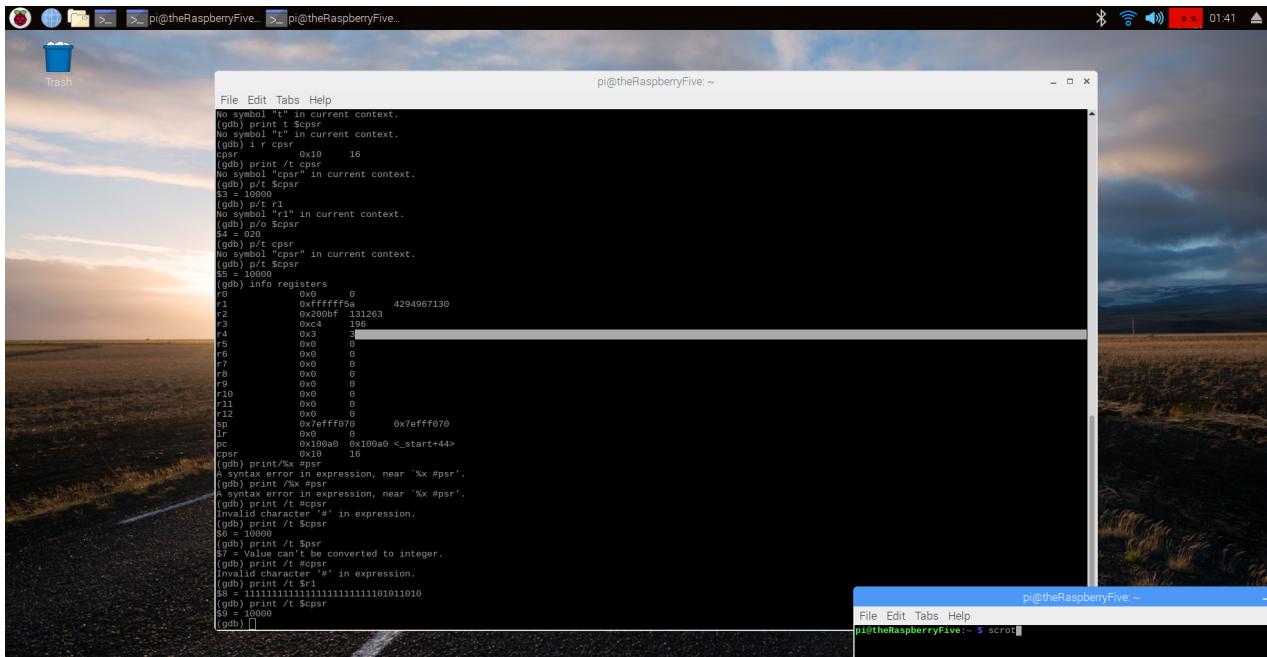
B) ARM Programming

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.



The screenshot shows a Raspberry Pi desktop environment with a terminal window open. The terminal window title is "pi@theRaspberryFive: ~". The content of the terminal is as follows:

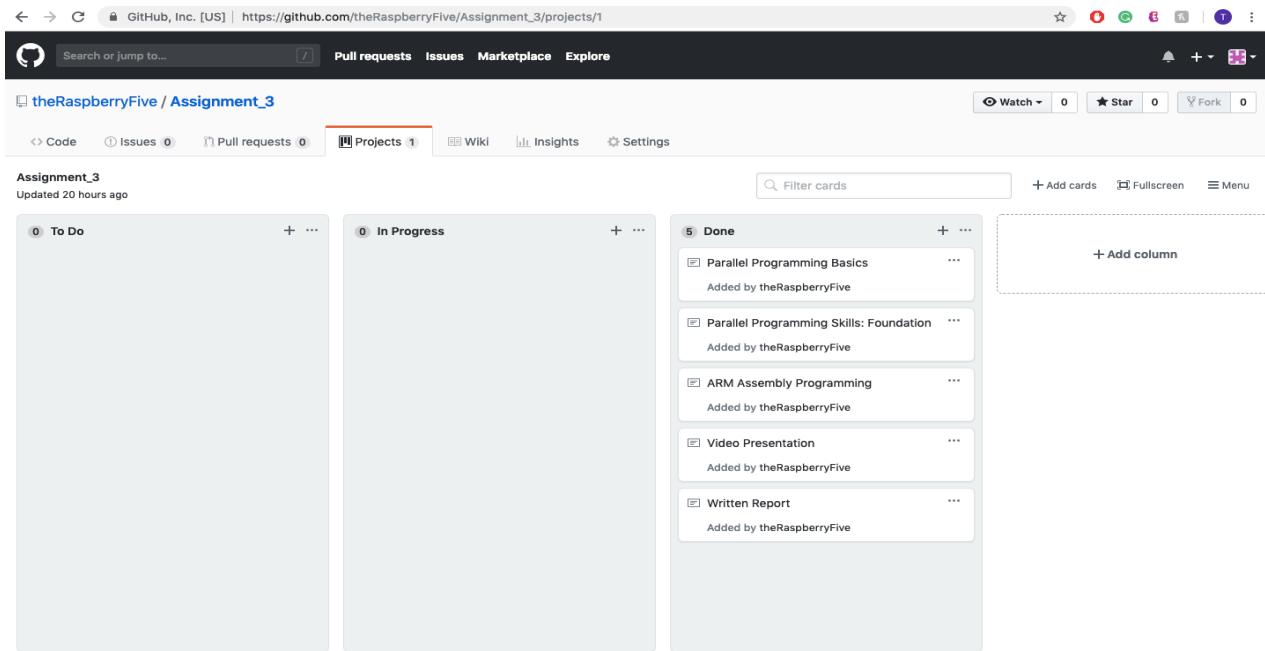
```
No symbol "t" in current context.
(gdb) print t $cpsr
No symbol "cpsr" in current context.
(gdb) l r cpsr
cpsr          0x10      16
(gdb) print /t cpsr
$3 = 10000
(gdb) p/t $cpsr
$3 = 10000
No symbol "r1" in current context.
(gdb) p/t $cpsr
$3 = 10000
No symbol "cpsr" in current context.
(gdb) p/t $cpsr
$3 = 10000
(gdb) info registers
r0            0x0      0
r1            0xfffffff4 4294967130
r2            0x200bf 131263
r3            0xc4    196
r4            0x3      3
r5            0x0      0
r6            0x0      0
r7            0x0      0
r8            0x0      0
r9            0x0      0
r10           0x0      0
r11           0x0      0
r12           0x0      0
sp            0x7efff070  0x7efff070
lr            0x0      0
pc            0x000000001000a0 <_start+44>
cpsr          0x10      16
(gdb) print/xx $psr
A syntax error in expression, near 'Xx #psr'.
(gdb) print /xx $psr
A syntax error in expression, near 'Xx #psr'.
A syntax error in expression, near 'Xx #psr'.
(gdb) print /t #cpsr
Invalid character '#' in expression.
(gdb) print /t $cpsr
$3 = 11111111111111111111111111010111010
(gdb) print /t $cpsr
$3 = 10000
(gdb) [ ]
```

The terminal prompt at the bottom right is "pi@theRaspberryFive:~ \$ scroll".

Task 5

Appendix

- <https://github.com/theRaspberryFive>



- <https://raspberrypive.slack.com/messages>

Task 6

Presentation

- <https://youtu.be/QP93TfhJCG8>