Developing Soft and Parallel Programming Skills Using Project-Based Learning

Spring-2019

Gitter-geeks

Hyoungjun Lee
Michael Knight
Tek Sharma
Vedansi Parsana – Team coordinator
Zoe Kosmicki

Section 1: Planning and Scheduling

As a team coordinator it was my responsibility to assign tasks to each and every member of my group fairly, it should not be given more to one and less to other. We used GitHub to upload the stuff we completed. We also used Slack to communicate. We decided a time which was appropriate for everyone to meet and record a video. We also helped each other with ongoing problems through the assignment. I was kind of difficult for me to assign task to everyone because as every member as to do task 3 and 4. So we coordinated and decided to give one member to edit video and one to assign to do tasks on GitHub. Overall I am happy that our group coordinated and performed really well throughout the project.

The following is a chart breaking down the individual contributions of our team members for this week's project:

Name	Email	Tasks	Dependency	Duration	Due Date	Note
Hyoungjun Lee	hlee113@student.gsu.edu	-Task 3 -Task 4 -Task 6 (Participation)	None	4 hours	3/5/20	100 %
Michael Knight	mknight23@student.gsu.edu	-Task 3 -Task 4 -Task 6 (Participation)	None	4 hours	3/5/20	100 %
Tek Acharya	tacharya2@student.gsu.edu	-Task 2, Github -Task 3 -Task 4 -Task 6 (Participation)	None	4 hours	3/5/20	100 %
Vedansi Parsana (Coordinator)	vparsana1@student.gsu.edu	-Task 1 -Task 3 -Task 4 -Task 5 -Task 6 (Participation)	Needs Everyone's submissions to combine into a report for submitting.	5 hours	3/5/20	100 %
Zoe Kosmicki	zkosmicki1@student.gsu.edu	-Task 3 -Task 4 -Task 6 (Participation), Video editing	None	4 hours	3/5/20	100 %

Hyoungjun Lee

TASK 3: Parallel programming skills

Foundation

Define the following: Task, Pipelining, Shared Memory, Communications, Synchronization. (in your own words)

Task : is an activity or works that you have to do, like part of project, in programming the task is a fundamental unit of programming that working frame work controls, operating system controls

Pipelining: process of accumulating and executing computer instructions, broken into several individual tasks by the processor.

Communication: how parallel data talk, exchange data each other

Shared Memory: part of random access memory, common memory location is shared between several processors through a bus, each processor has the same access to memory

Synchronization: process that coordinating two or more activities, devices or processes in same time.

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

Flynn's taxonomy: specific classification of parallel computer architectures based on number of concurrent instruction

(SISD) single instruction, single data : can only process and execute one instruction or data stream at a time

(MISD) multiple instruction, single data : parallel computer, excute more than one set of instruction, processing unit operates on the data independently via separate instruction using single data.

(SIMD) single instruction, multiple data : parallel computer, execute same instruction to various data thread at same time

(MIMD) multiple instruction, multiple data : parallel computer, every processor will be running into different instruction executes multiple instruction and data streams at the same time

What are the Parallel Programming Model

Shared memory, threads, distributed memory, data parallel, hybrid, SPMD, and MPMD

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

UMA(Uniform Memory Access) and NUMA (Non-Uniform Memory Access) , these are the types of parallel computer memory architectures, UMA have 1 shared general memory that can access multiple CPU, NUMA have several memories that connected to several CPU, OpenMP can be used both because it designed to multi processore, and shared memory machine.

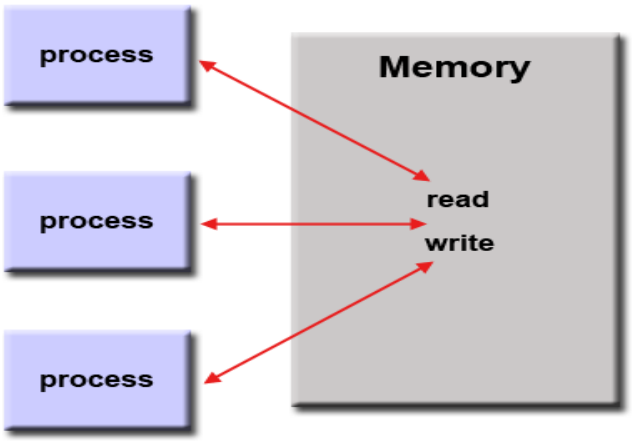
Compare Shared Memory Model with Threads Model? (in your own words and show pictures

Shared memory model : memory is shared, each process can read and write to share address inside memory as shown in the picture, and this is basic type of memory model SMM is hard to manage data while implementing

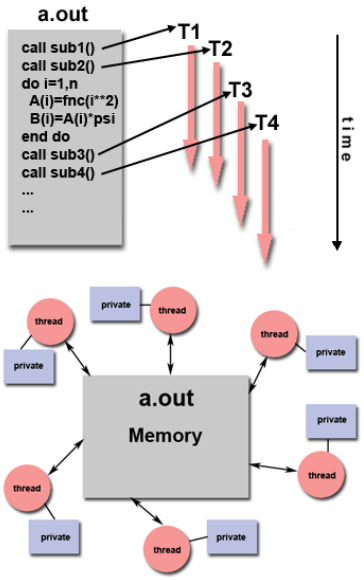
Threaded Memory Model (**TMM**)

Program has a main body, single thread process can have multiple ways to execute, while implement, using OpenMP

Shared memory



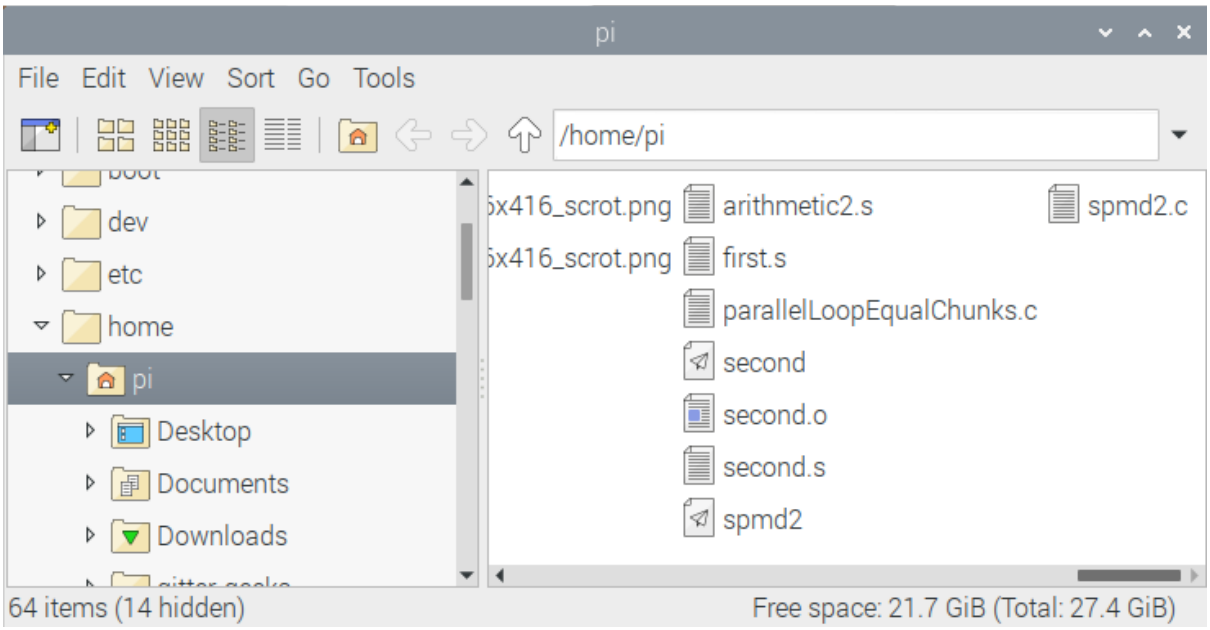
Threaded memory



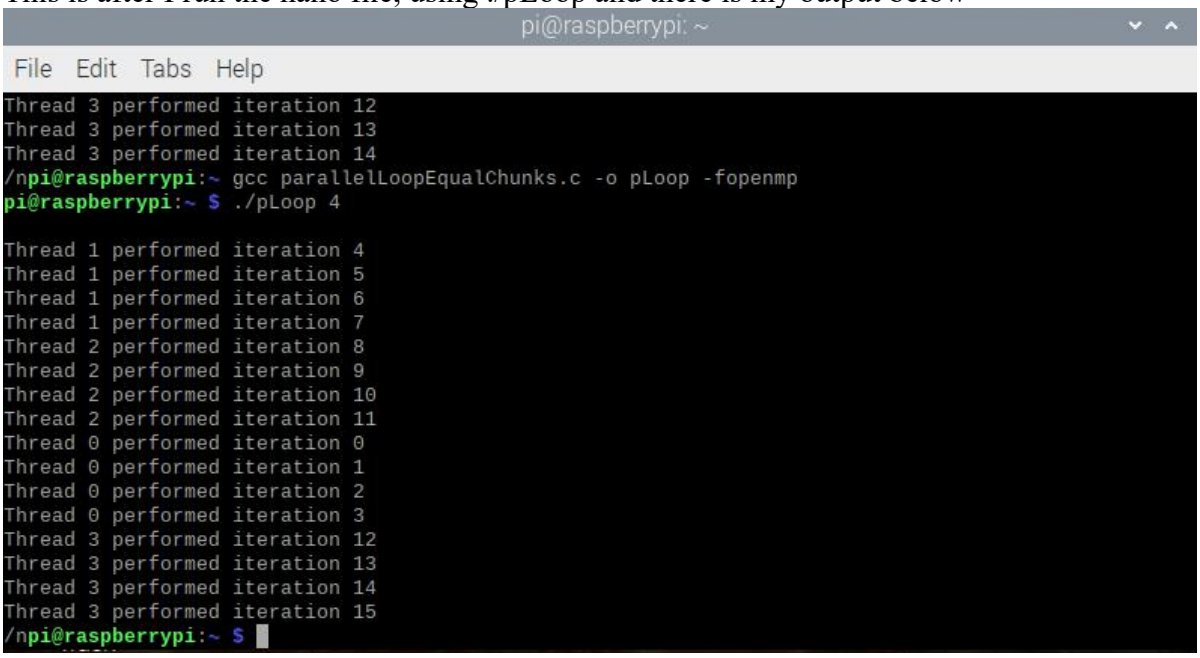
What is Parallel Programming:
A programming technique that is used to perform several computations or execution simultaneously, better option to complete task than serially
What is system on chip(SoC)?
This is a type of computer has the CPU, GPU, and RAM put onto a single chip,

Does Raspberry PI use system on SoC?
Yes
Explain what the advantages are of having a System on a Chip rather than separate CPU, GPU and RAM components
1 most advantage thing is reducing its size, refers it will be portable and less expensive, and better for smaller devices.

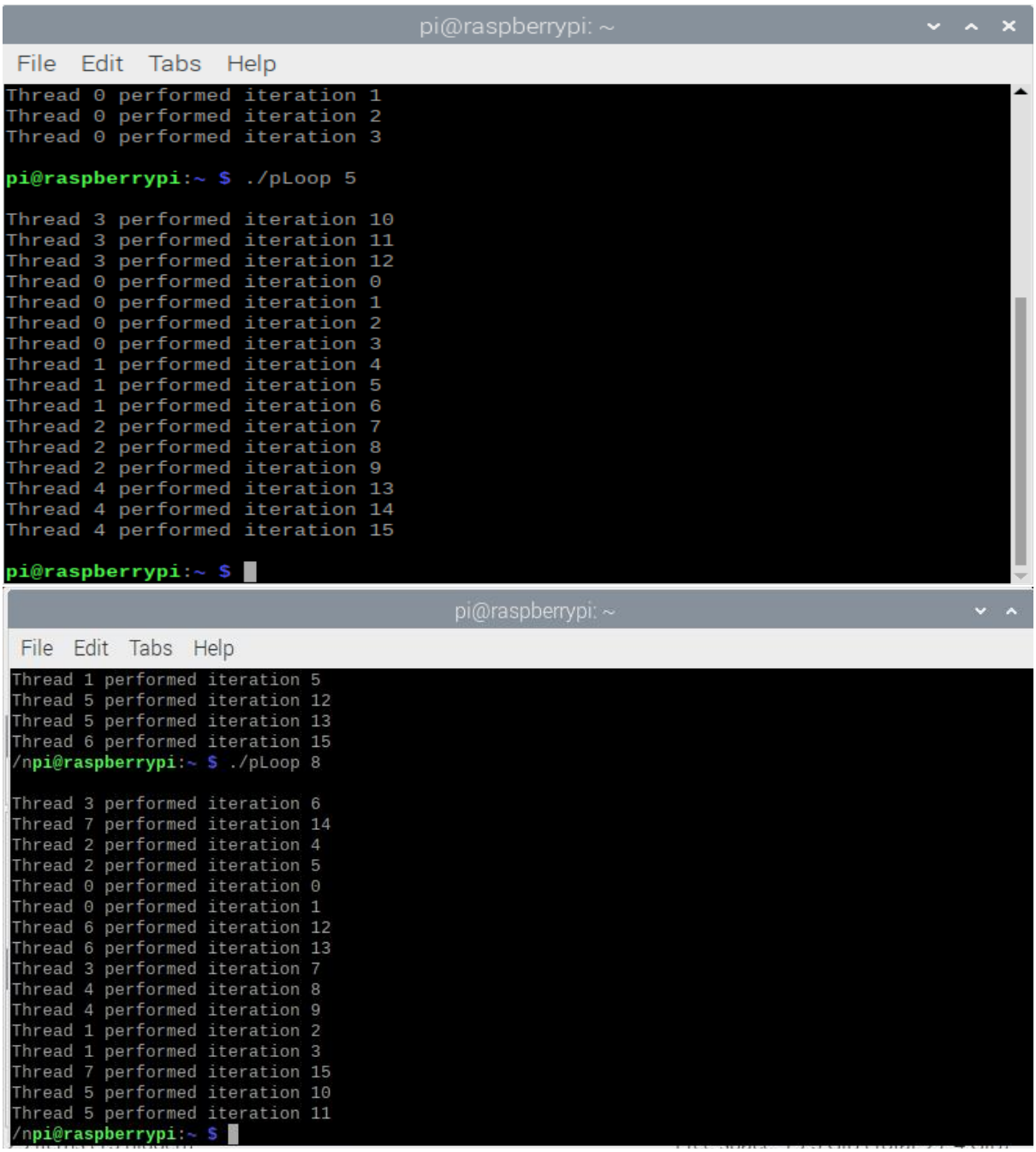
40p) Parallel Programming Basics



This is after I run the nano file, using ./pLoop and there is my output below

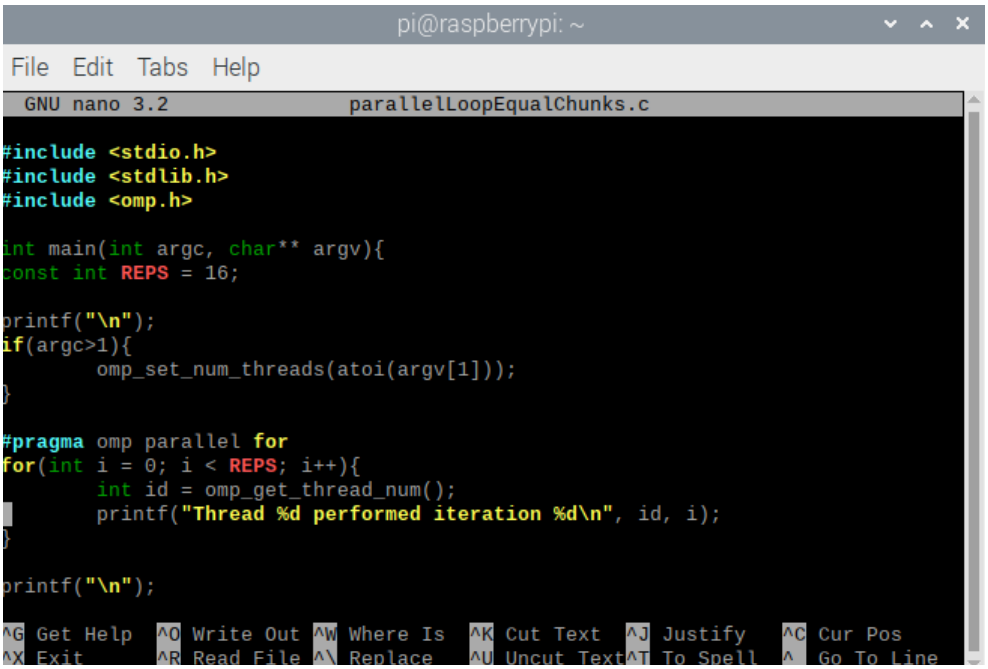


Also I tried more times to see how does pattern and output looks like

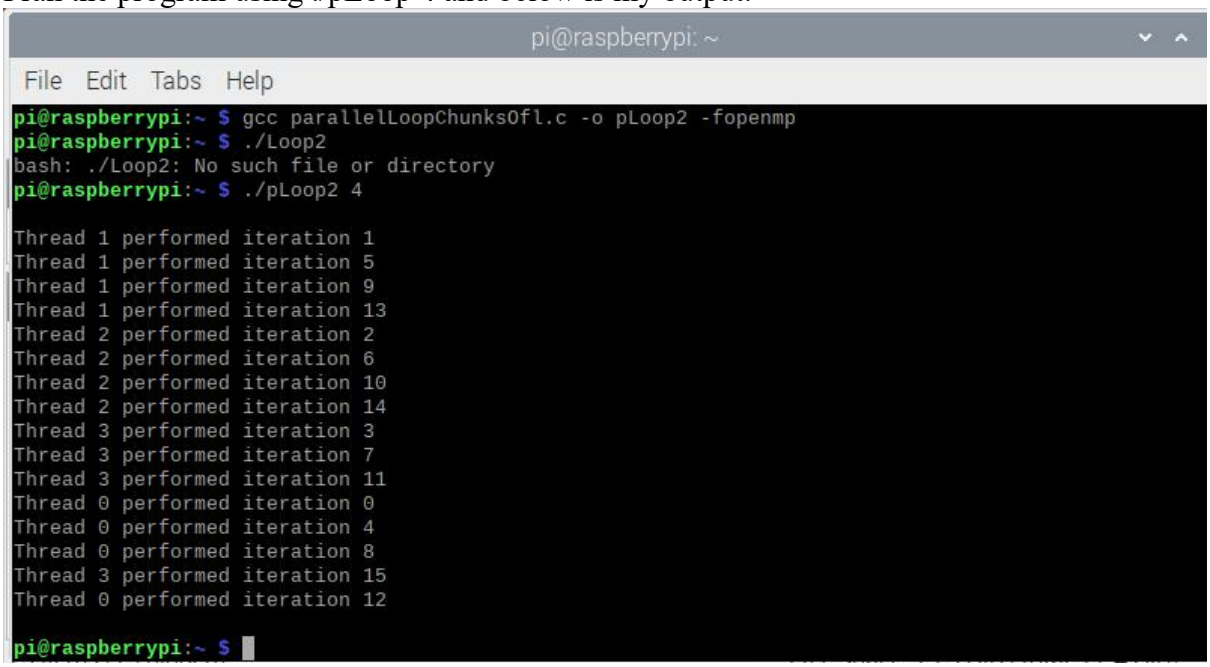


After watching some patterns, if you put number after ./pLoop, the thread will execute number between (0~number -1)

And there is an alternative way to divide an work,
I wrote parallelLoopChunksOf1.c



I ran the program using ./pLoop 4 and below is my output.



I think its not quite distributed evenly, and used static scheduling system.
Output is quite different compare to first one
This is like different order but same output

3.4 Dynamic scheduling for time-varying task

I wrote reduction.c by using nano program, and I compiled it and ran it,

```
pi@raspberrypi:~ $ nano reduction.c
pi@raspberrypi:~ $ gcc reduction.c -o reduction -fopenmp
pi@raspberrypi:~ $ ./reduction

Sequential sum:         499562283
Parallel sum:   499562283

pi@raspberrypi:~ $ ./reduction 4

Sequential sum:         499562283
Parallel sum:   499562283

pi@raspberrypi:~ $ ./reduction 7

Sequential sum:         499562283
Parallel sum:   499562283

pi@raspberrypi:~ $
```

I put iteration input as 4 first, both parallel sum and sequential sum are the same,

In order to see different output, we uncommented the first comment and see the result
I put both output for commented and uncommented

```
pi@raspberrypi:~ $ nano reduction.c
pi@raspberrypi:~ $ gcc reduction.c -o reduction -fopenmp
pi@raspberrypi:~ $ ./reduction 4

Sequential sum:         499562283
Parallel sum:   151974691

pi@raspberrypi:~ $ nano reduction.c
pi@raspberrypi:~ $ gcc reduction.c -o reduction -fopenmp
pi@raspberrypi:~ $ ./reduction 4

Sequential sum:         499562283
Parallel sum:   499562283

pi@raspberrypi:~ $
```

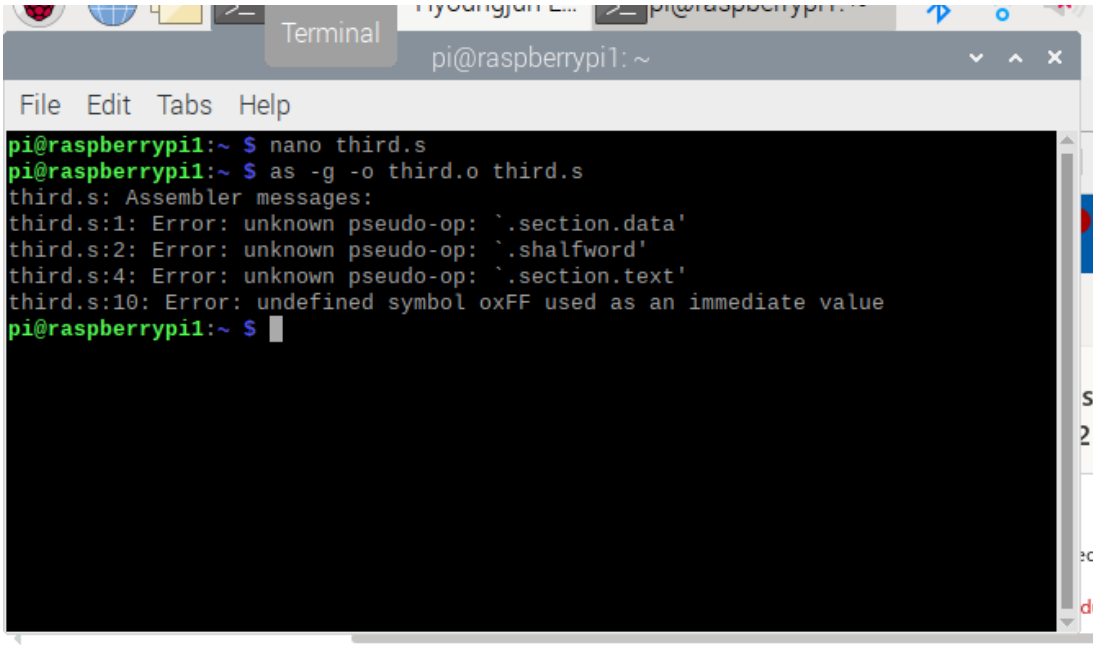
Output result
Sequential sum = 499562283
Parallel sum = 151974691

TASK 4: Arm assembly programming.

Task A)

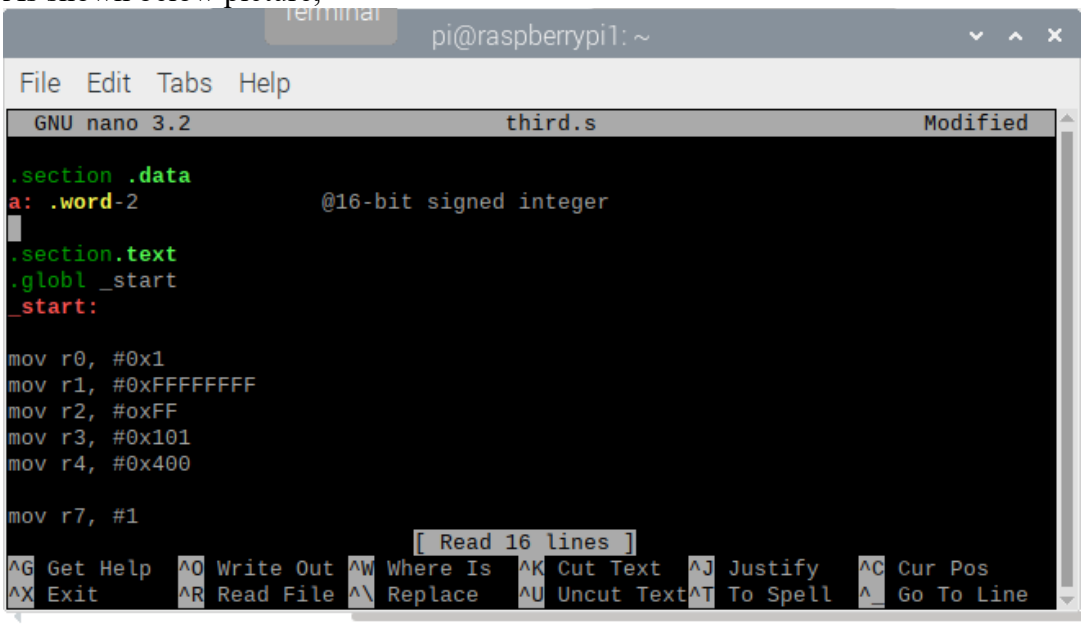
- What error message did you get and why? (Report that)

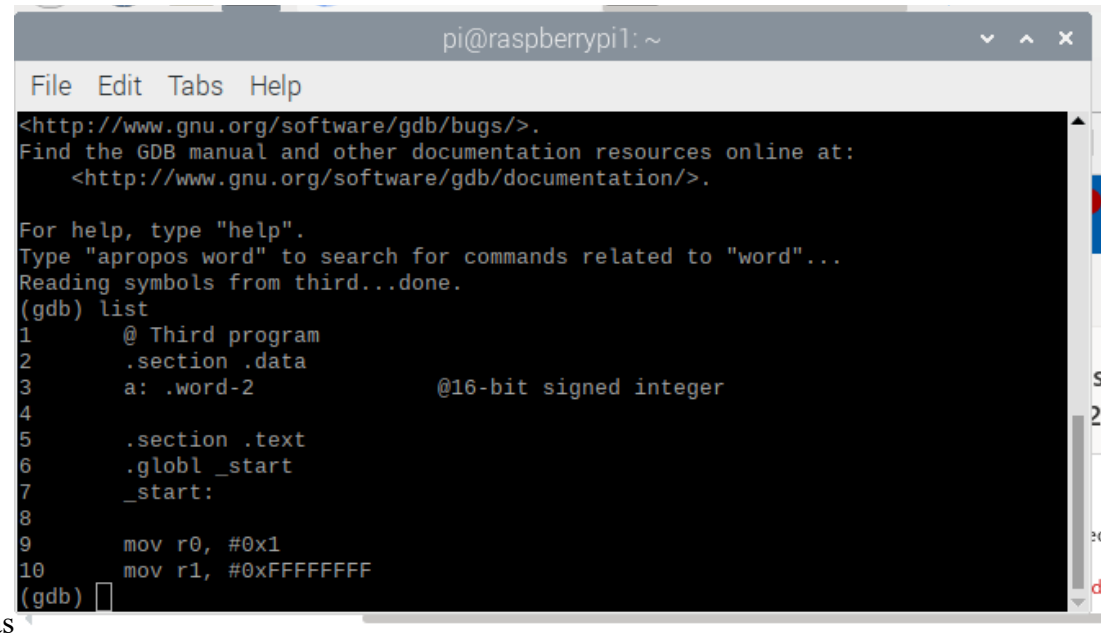
We found .shalfword error in the third.s
Why?
Shalfword cannot be a data declaration



after correction the declaration variable type to .word, the code assembled correctly

As shown below picture,

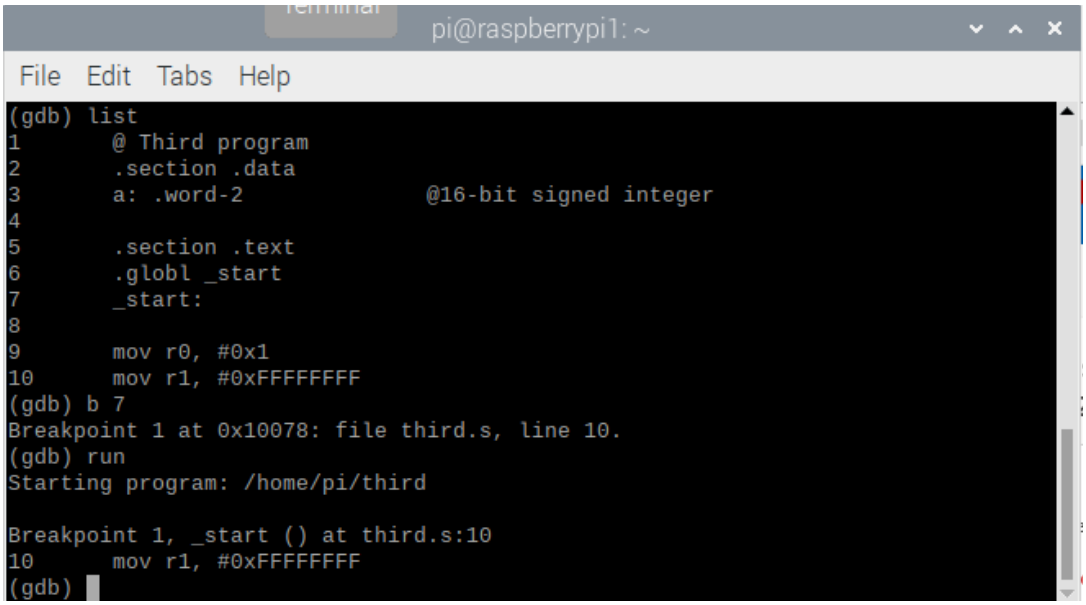


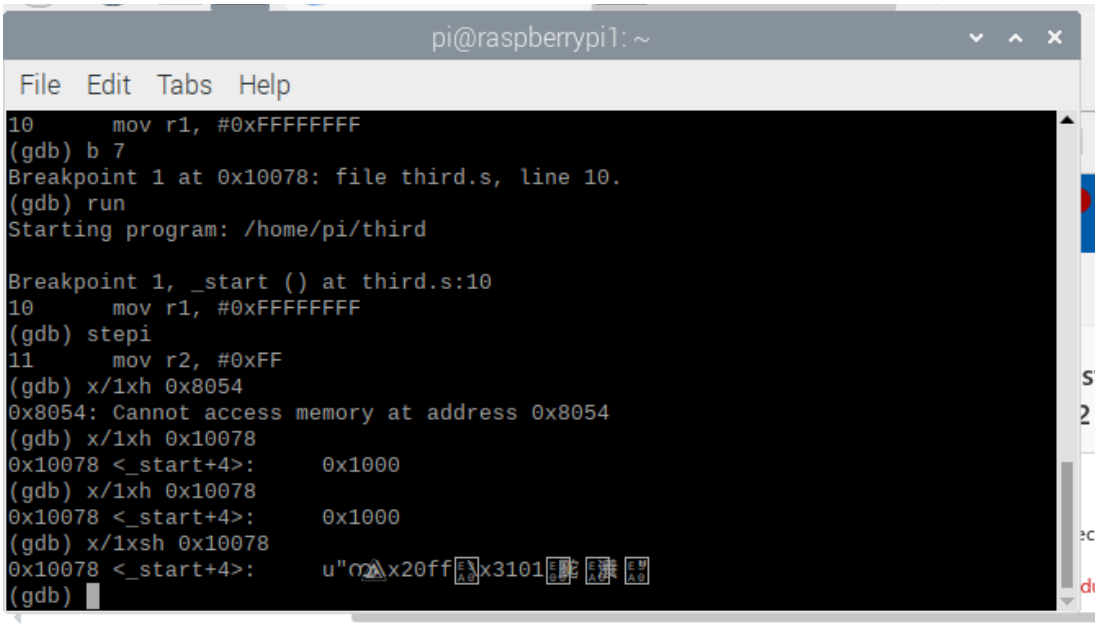


as

and we set up the break point at line 7, but it moved to line 10 automatically, we tried to test register or memory by using (gdb) x/1xh 0x10078.

Pictures below



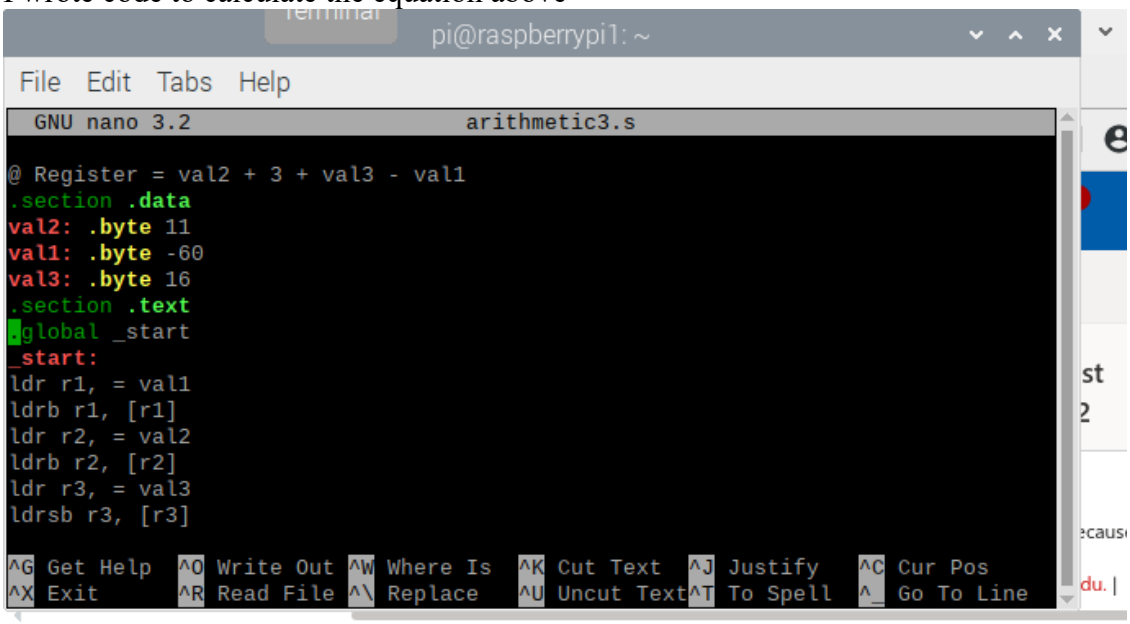


With xh(unsigned) and xsh(signed), we got same offset <_start+4> but for the address, I cannot define sign value's address

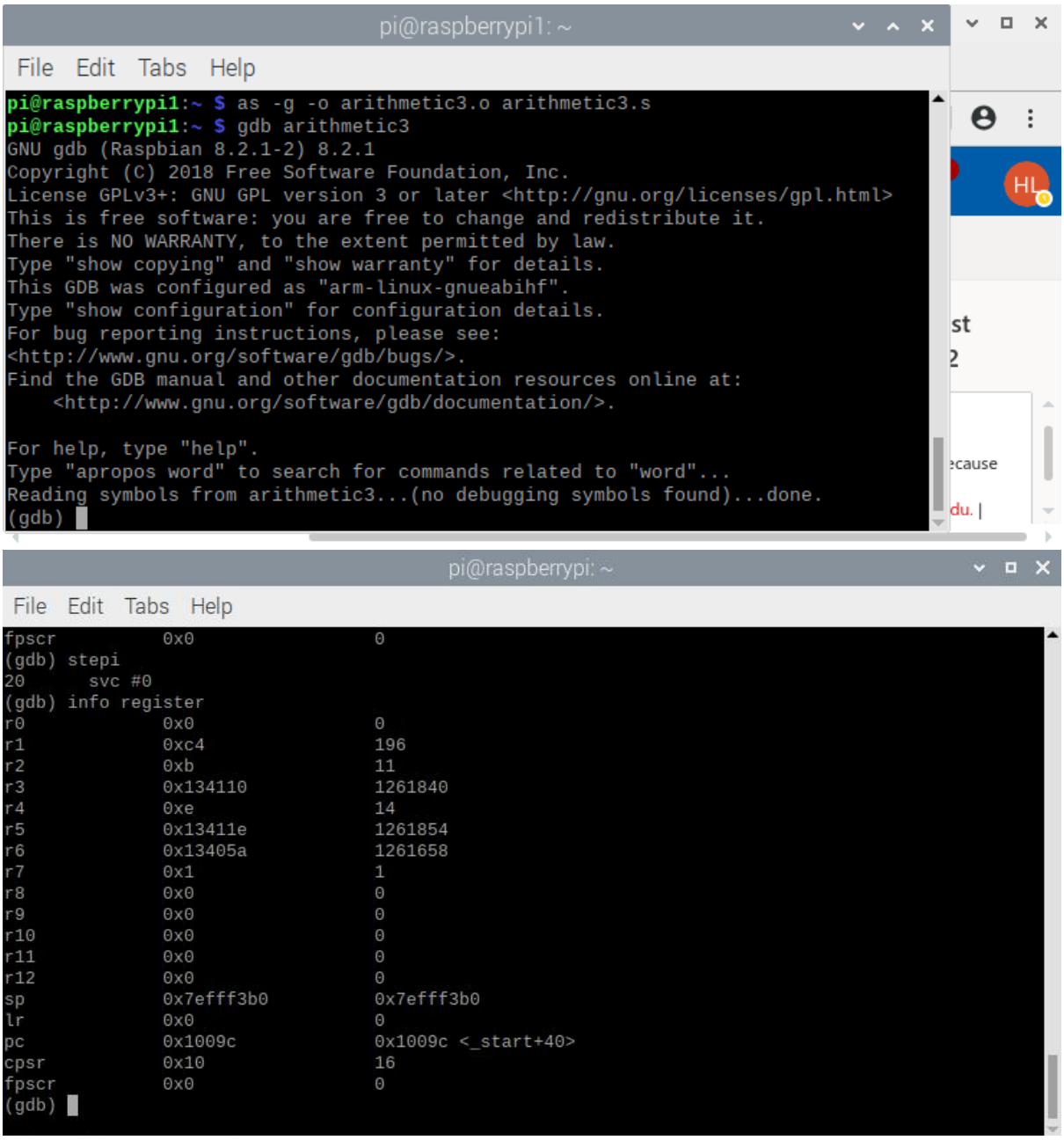
Task B)

Register = val2 + 3 + val3 – val1; val1 = -60(signed), val2 = 11(unsigned), and val3 = 16(unsigned) and all these memories are 8-bit integer memory

I wrote code to calculate the equation above



Then we assembled and linked together and ran it, picture is below



With the assigned value, we know register would be 90

11+3+16-(-60) = 90.

r1 = C4h which corresponds to -60 (11000100) that is our Val1

r2 = 0Bh which corresponds to 11 that is our val2

r3 = 10h which corresponds to 16 that is our val3

similarly

r4 = 14

r5 = 30

r6 = 90 (5a) which is our final answer of the “Register”

when we check “cpsr” value, negative value is set

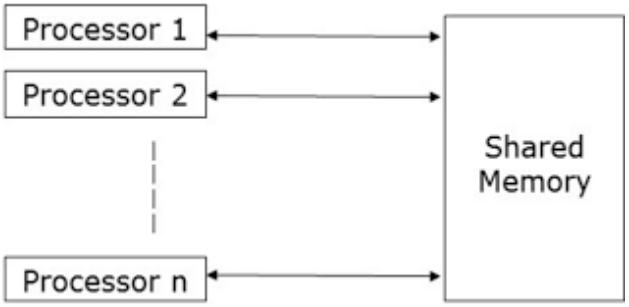
all these values are expected.

Michael Knight

TASK 3: Parallel programming skills

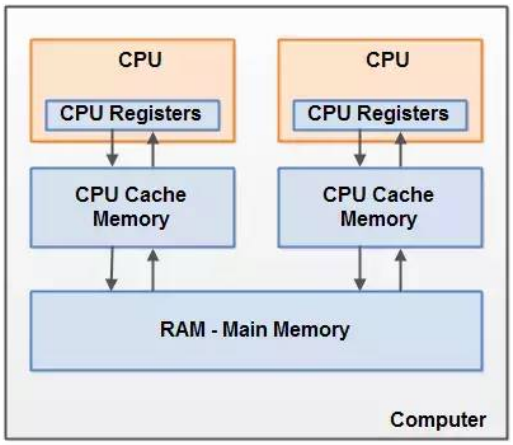
Defined Terms:
Task – Is instructions given to the computer that the carried out by the processor.
Pipelining – Is a way to break down a task for a processor to use its multiple cores to carry out the task.
Shared Memory – Memory that is shared between all the processors.
Communications – Refers to the data that is exchanged in parallel programming.
Synchronization – Process that coordinates running at the same time or waiting until the rest is finished.
Classify parallel computers based on Flynn's taxonomy:
SISD – Single instruction stream, Single data stream. One instruction, one data, oldest type of computer, Deterministic execution
SIMD – Single instruction steam, multiple data stream. All processing units execute the same instructions at any given clock cycle, each processing unit can operate on a different data element. Best suited for specialized problems characterized by a high degree of regularity.
MISD – Multiple instruction stream, single data stream. Each processing unit operates on data independently. Single data stream is fed into multiple processing units. Very few of these systems ever existed if any.
MIMD – Multiple instruction stream, multiple data stream. Every processor may be executing a different instruction stream; every processor may be working with a different data stream. Most common type of parallel computer.
What are the Parallel Programming Models:
Shared memory, threads, distributed memory, data parallel, hybrid, SPMD, and MPMD
List and briefly describe the types of Parallel Computer Memory Architectures. What type is used by OpenMP and why:
UMA – Uniform Memory Access. Share 1 general memory that can access multiple CPU
NUMA – Non-Uniform Memory Access. Several memories that are connected to several CPU
OpenMP was made for multi-core systems, both UMA and NUMA is used by OpenMP.
Compare Shared Memory Model with Threads Model:

Shared Memory Model:



Each Processor has access to the shared memory which that can operate from.

Threaded Memory Model:



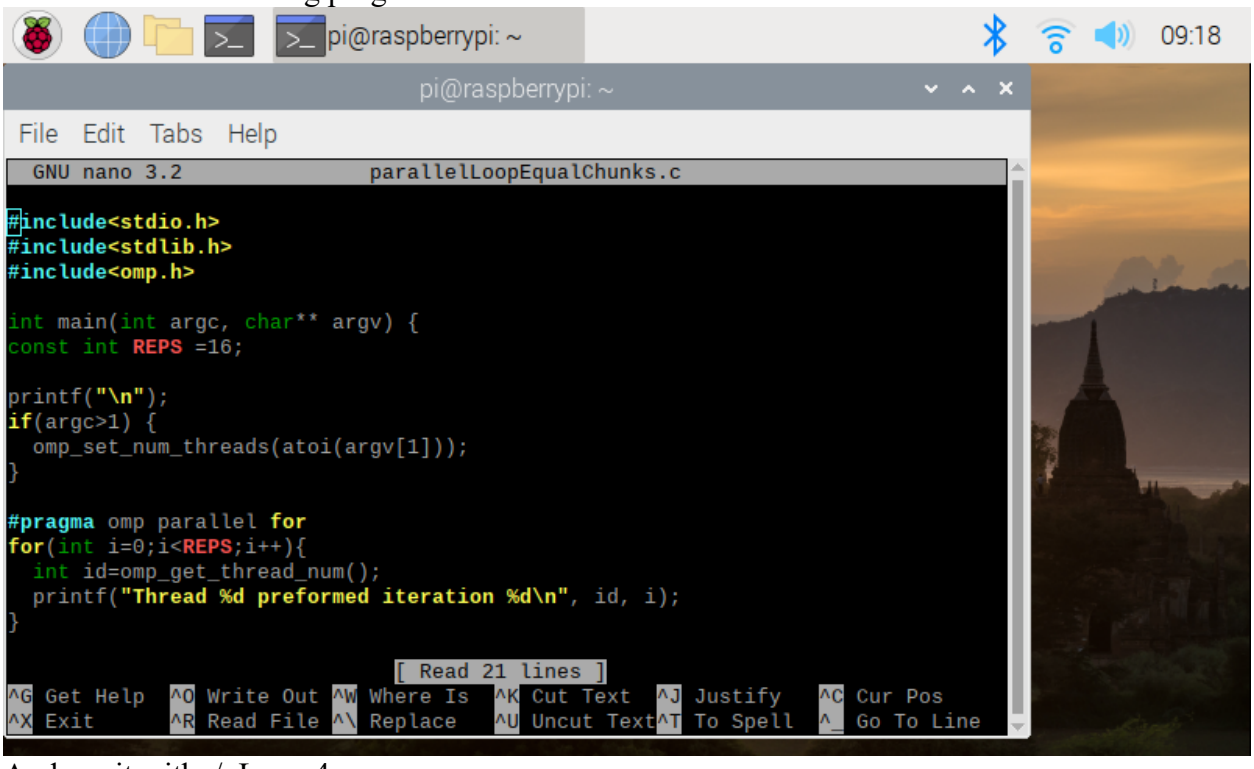
Each Processor has access to their own memory that is used for them to carry out the task, while that "private" memory is connected to the shared memory so it can be updated with the results.

What is Parallel Programming?
Programming that brings out the most potential that's possible out of a computer and all of its cores to reduce the amount of downtime or clockcycles.

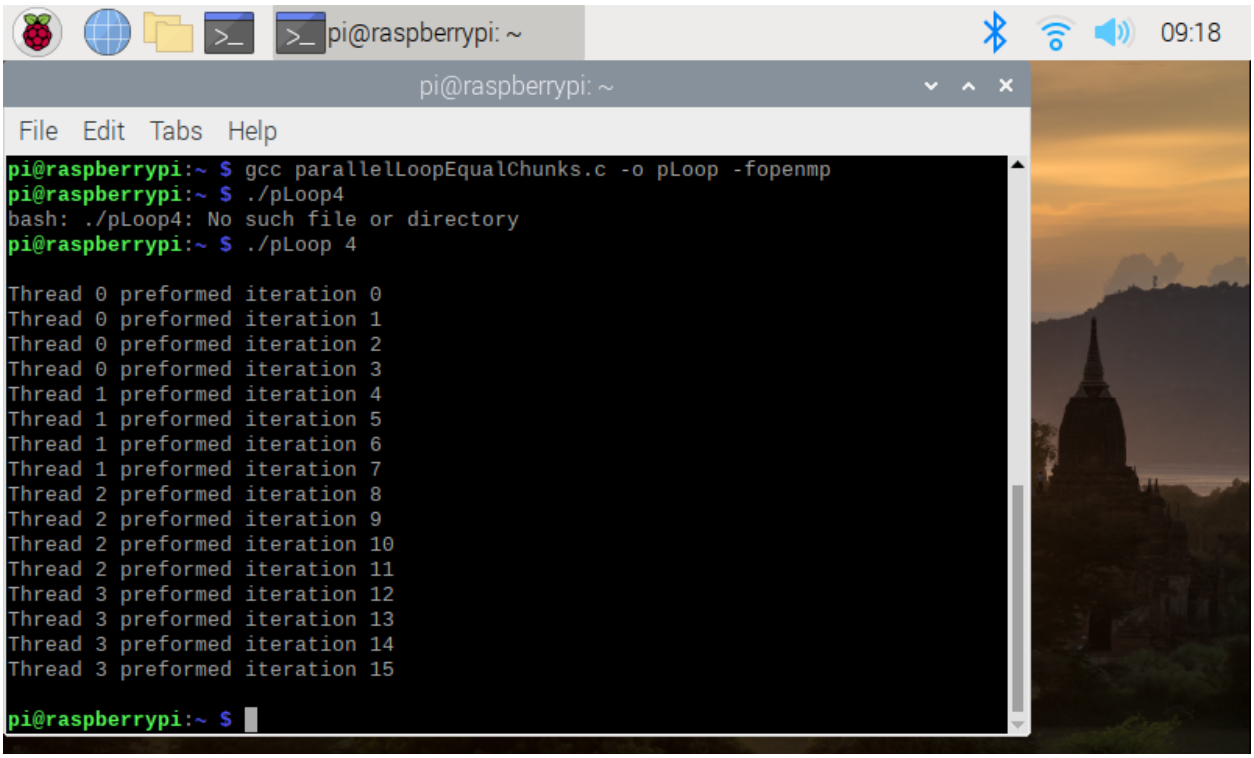
What is system on chip(SoC)? Does Raspberry PI use system on SoC?
It's a chip that has the CPU, GPU, and Ram all on the same chip. Yes Raspberry PI Uses this.

Explain what the advantages are of having a System on a Chip rather than separate CPU, GPU and RAM components.
Its more portable due to its size being smaller, and less expensive.

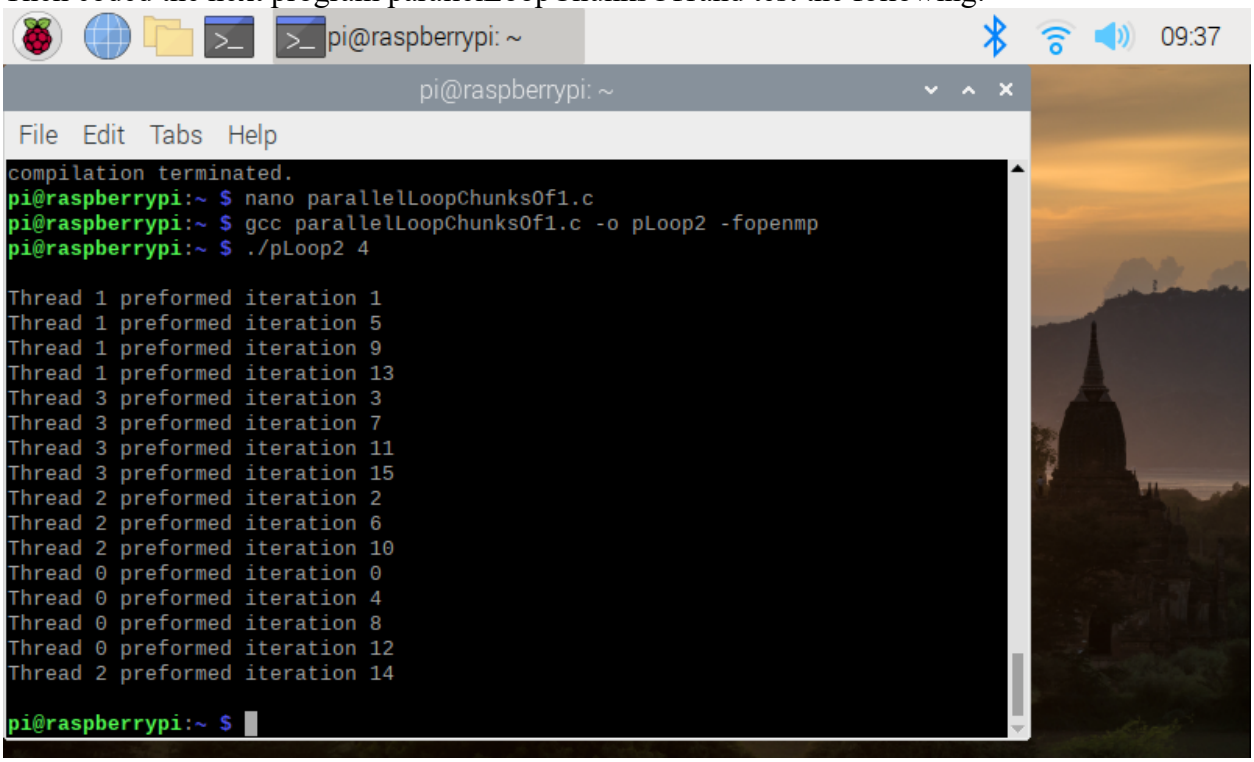
Parallel Programming Basics
2.1 : Made the following program



And ran it with ./pLoop 4:



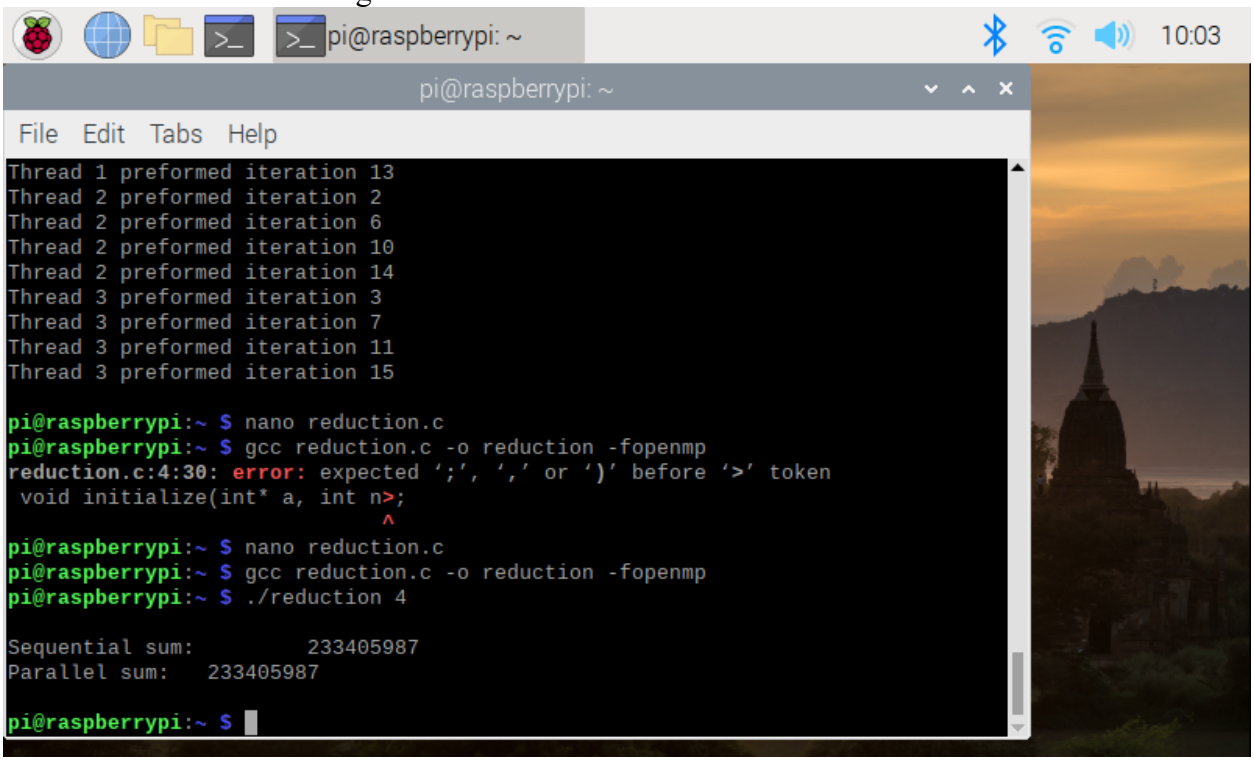
Then coded the next program parallelLoopChunksOf1and test the following:



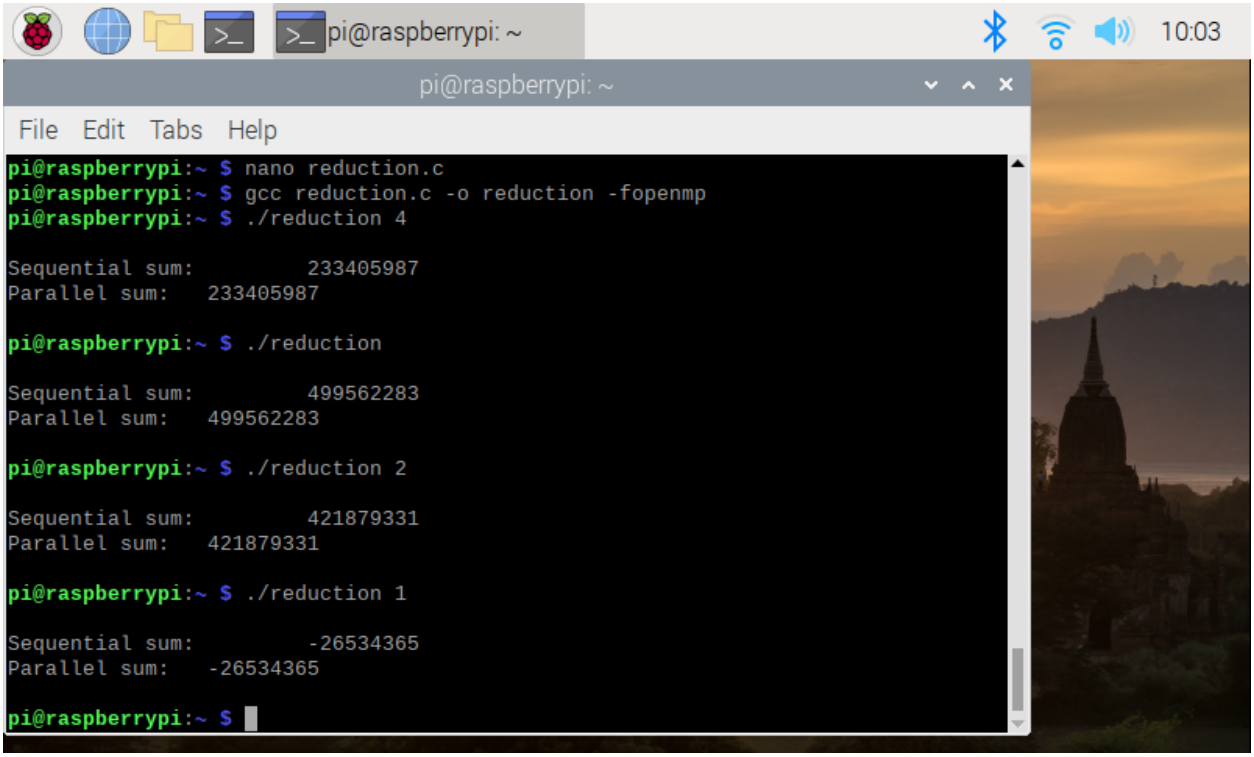
After uncommenting the section of code and running it again, I got the same result but in a different order.

Next was instructed to write the program Reduction.c

I was met with the following result

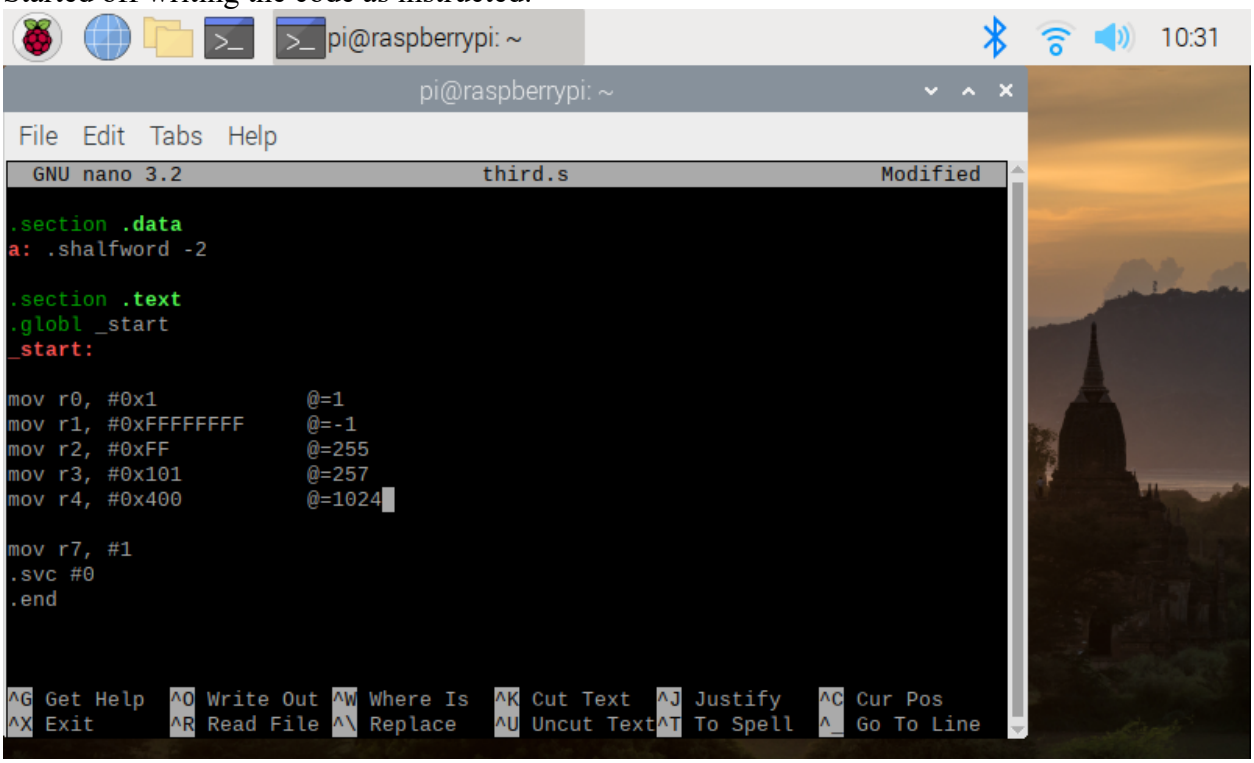


I then ran different numbers threw it:

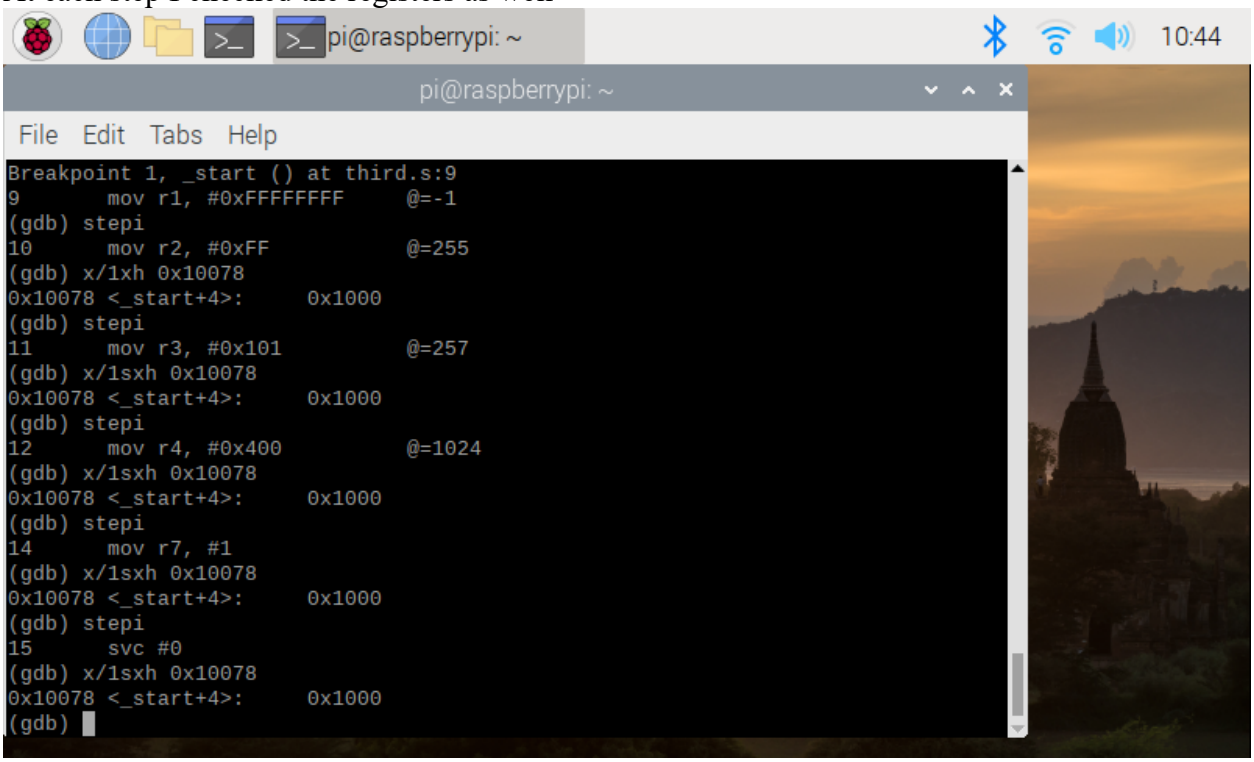


TASK 4: Arm assembly programming.

Started off writing the code as instructed:



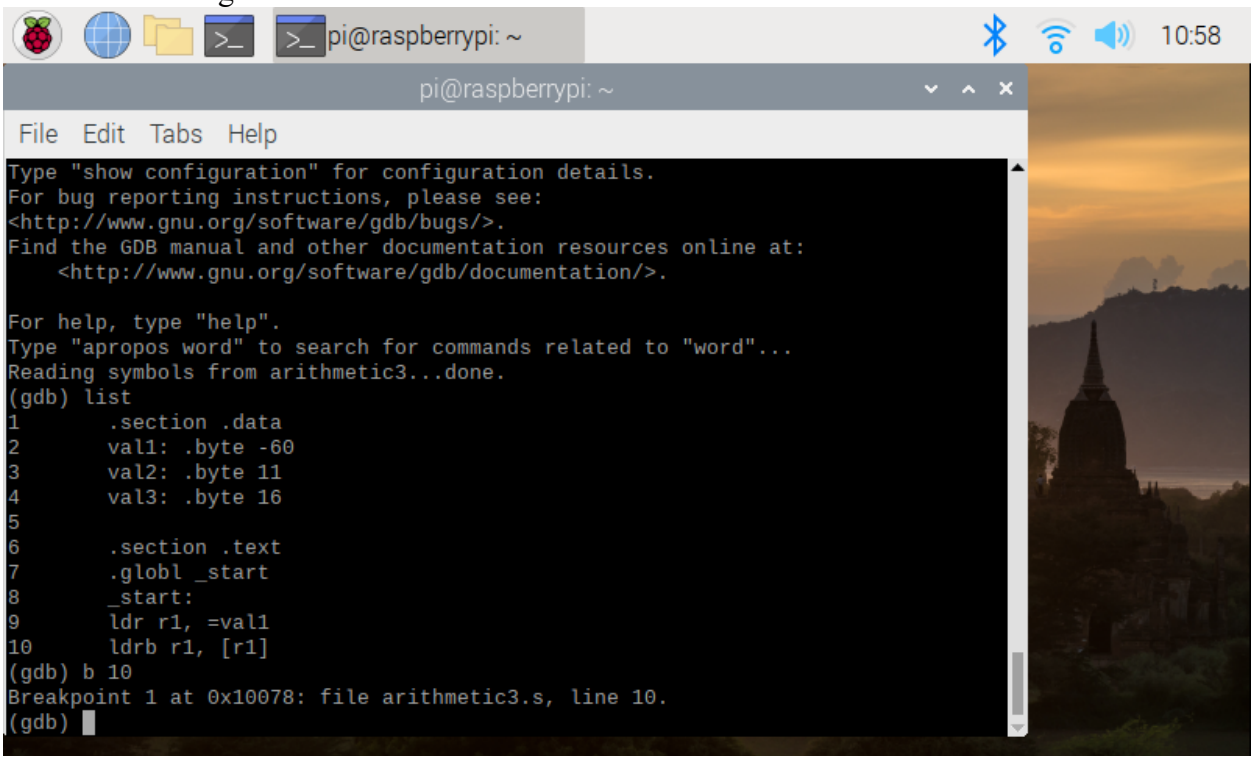
Got an error message that the complier didn't understand ".shalfword" and this is because it should be ".hword" instead.
Then compiled and ran dbug
At each step I checked the registers as well



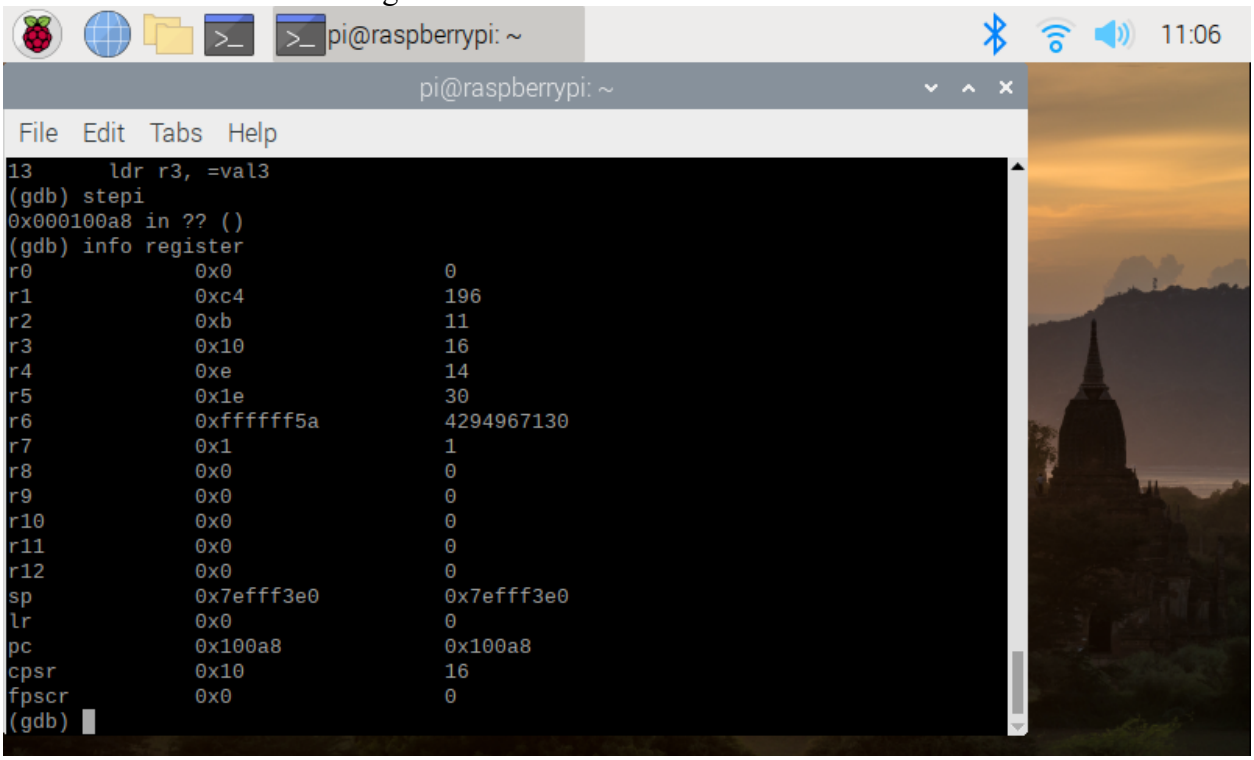
Part2:

Instructed to write the program Arithmetic3.s:

This is it in debug mode:



And this is the result of the registers after its ran



Tek Acharya

TASK 3: Parallel programming skills

1. **Define the flowing: Task, Pipelining, Shared Memory, Communications, Synchronization (in your own words) (5P)**

Task

A task is a set of programs/instructions that a processor can execute. In other words, any executable file to the microprocessor is a task to it.

Pipelining

Pipelining is a type of parallel computation in which a task is broken into several individual tasks by the processor by streaming the inputs, much like an assembly line.

Shared Memory

A shared memory, from a hardware point of view, is a computer architecture in which a common memory location is shared between several processors through a bus. And in a programming point of view shared memory is a common data reservoir accessing through instructions regardless of where the memory is located.

Communication

The exchange of data during parallel computation is commonly referred to as communication.

Synchronization

Closely associated with communication, the collaboration of parallel tasks in real-time, is called synchronization. This is implemented by generated by creating a synchronization point. Often next task many not proceed until one is done creating a wait increasing the wall clock execution.

2. **Classify parallel computers based on Flynn's taxonomy. Briefly describe every one of them (8P)**

The classification of parallel computers based on Flynn's taxonomy is briefly described below.

A. Single Instruction stream, Double Data stream (**SISD**):

A very classic non-parallel computer in which single instruction is acted on by CPU during a clock cycle that uses single data as its input. This type is basically the first-generation computer which is very deterministic in execution. Example of such computer type includes minicomputers, workstations, and single-processor/cores Pcs.

B. Single Instruction, Multiple Data (**SIMD**):

A basic type of parallel computer which uses single instruction but can execute multiple data element at any clock cycle. Most suited for graphics processors units (GPU), this type comes with two varieties: Processor Array and Vector Pipelines. An example includes Thinking Machines CM-2, IBM 9000, Hitachi S820, etc.

C. Multiple Instruction, Single Data (**MISD**):

This also is a type of parallel computer in which each processing unit operates on the data independently via separate instruction using single data. This type of computer system is rare, it is believed that the malware programmer uses this type trying to decrypt the information to heck.

D. Multiple Instruction, Multiple Data (**MIMD**):

This is also a type of parallel computer which is the most modern. This type of computer architecture, every individual processor will be running a different instruction using their own data stream. This type of system works synchronous or asynchronous, deterministic or non-deterministic. Examples include intel core i7, supercomputers.

3. What are the parallel Programming Models? (7P)

There are several parallel programming models in common use. Some of them are:

I) Shared Memory (without thread)
II) Threads
III) Distributed Memory / Message Passing
IV) Data-Parallel
V) Hybrid
VI) Single Program Multiple Data (**SPMD**)
VII) Multiple Program Multiple Data (**MPMD**)

4. List and briefly describe the types of Parallel Computer Memory Architectures. What type is used in OpenMP and why? (12P)

The list along with a brief description of Parallel Computer Memory Architecture is done bellow.

A. Uniform Memory Access (**UMA**)

This type of parallel memory architecture represented mostly by Symmetric Multiprocessor (**SMP**)machines with identical processors. In this architecture, each execution uses equal access and access time to the common memory location. This type of architecture also uses a coherent cache memory system in which an update on any one of the processors leads to update to the rest of the processors. The term is often called as **CC-UMA** (Cache Coherent UMA). This way processors remain vigilant at each time an execution occurs.

B. Non-Uniform Memory Access (**NUMA**)

This type of architecture is often made by physically linking two or more SMPs where one SMP can access the other one directly, though it is a slower method. Not all processors have equal access time to all memory thus minimizing the error. If Cache coherency is maintained, then may also be called **CC-NUMA**; Cache Coherent NUMA

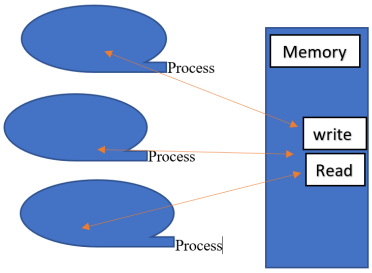
⇨ Advantages:
- Global address space provides a user-friendly programming perspective to memory.
- Data sharing between tasks is both fast and uniform due to the proximity of memory to CPUs.

⇨ Disadvantages:
- The lack of scalability between memory and CPUs. Adding more CPUs increases traffic to the signal transfer thereby creating performance reliability.
- Programmers are more responsible for correct access to global memory.

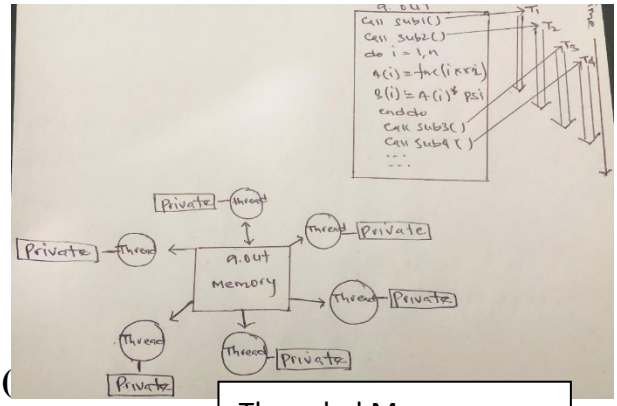
OpenMP is designed for multi-processor/core, shared memory machines. The underlying architecture can be shared memory UMA or NUMA. Because OpenMP is designed for shared memory parallel programming, it largely limited to single-node parallelism

5. **Compare Shared Memory Model with Thread Model (in your own word and show pictures) (10P)**

Shared Memory Model (**SMM**)	Threaded Memory Model (**TMM**)
This programming model shares a common address space to read and write to.	This model is a special type of SMM
This is a basic type of memory model	In this type, a single thread process can have multiple execution paths.
Difficult to understand and manage data locality while implementing	The implementation using OpenMP is user-friendly and potentially fewer errors.



Shared Memory



Threaded Memory

6. **What is parallel programming (**

A programming technique that is used to perform several computations or execution simultaneously. In this programming, several tasks are performed at a time. This is possible due to the involvement of multiple processors/cores that are used concurrently. This method has met the demand of modern computational need. It is fast and more accurate while using this technique.

7. **What is a system on chip (SoC)? Does Raspberry PI use the system on SoC? (5P)**

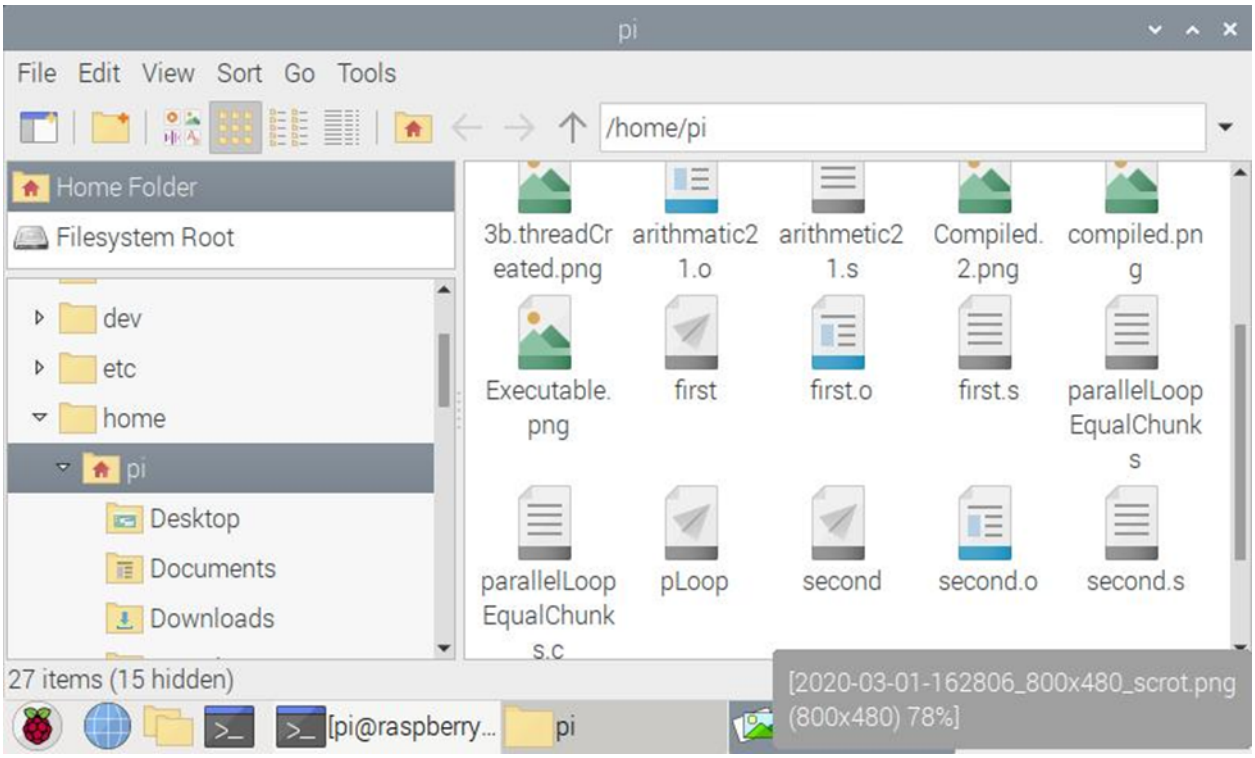
System on chip is a modern approach of reducing the size of a computer by integrating CPU (Central Processing Unit), RAM (Random access memory), or GPU (Graphics Processing Unit) and many other components essentially an entire computer into a single chip. Yes, the Raspberry PI is a SoC

8. **Explain what the advantages are of having a system on chip rather than having separate CPU, GPU and RAM components(5P)**

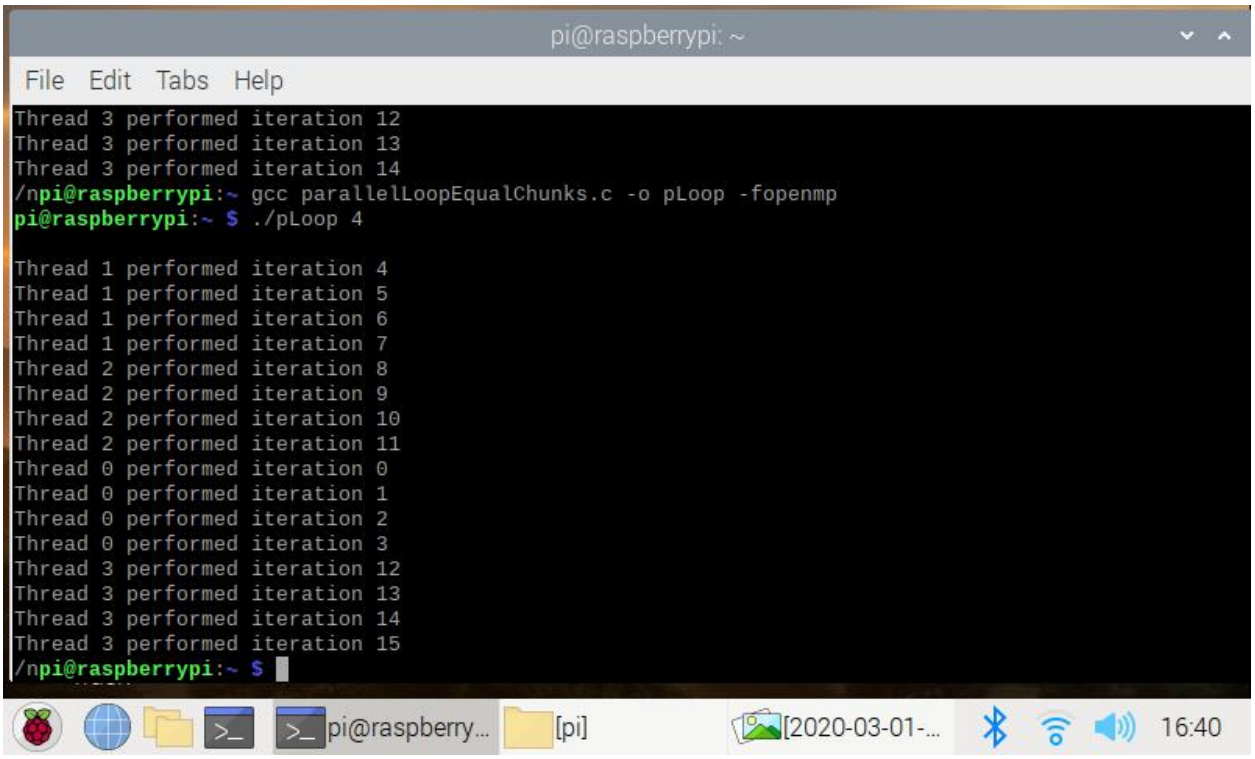
The most advantage of having SoC is its highly reduced size. That means portability and less expensive. Due to the high level of integration and much shorter wiring, an SoC uses considerably less power. Also, mobile computing will hike as everything will be handheld. The rate at which data are processed is also significantly increased due to this integration.

TASK3b. Parallel Programming Basics A3(80P)

What happens when the number of iterations (16 in this code) is not divisible by the number of threads?

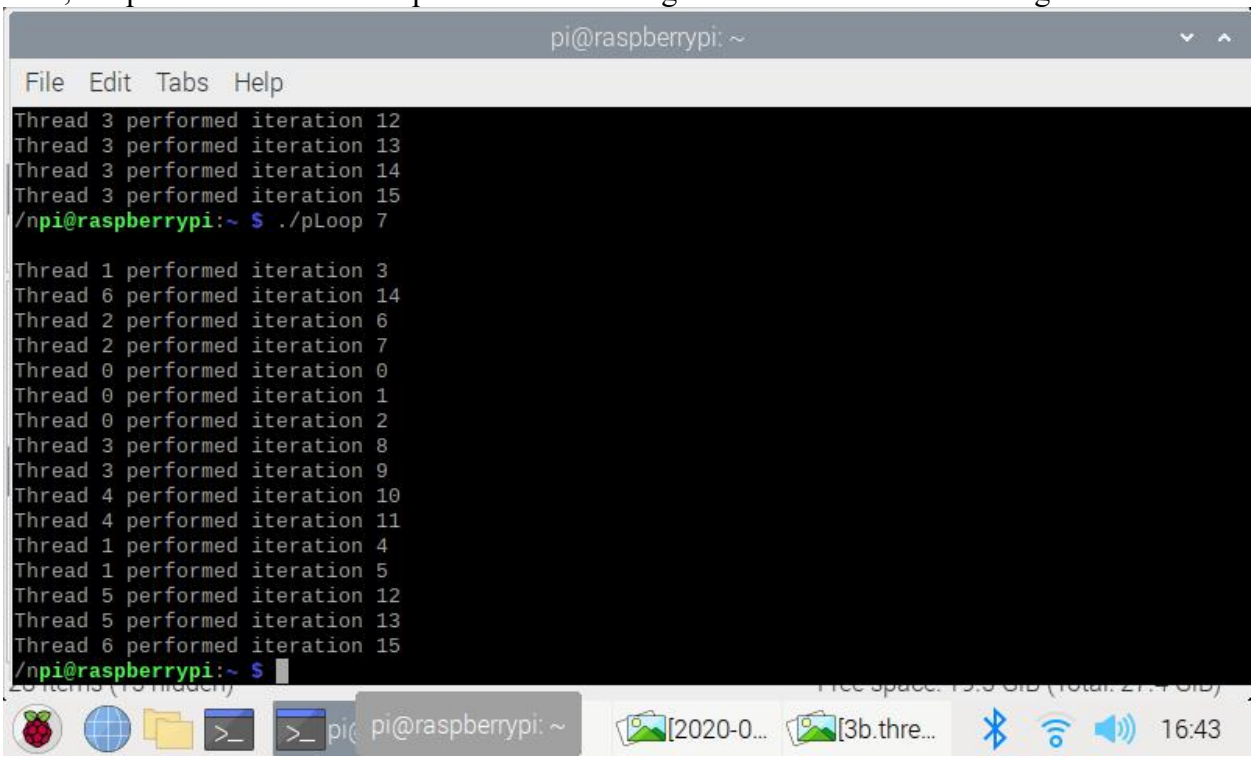


The executable pLoop file is created.

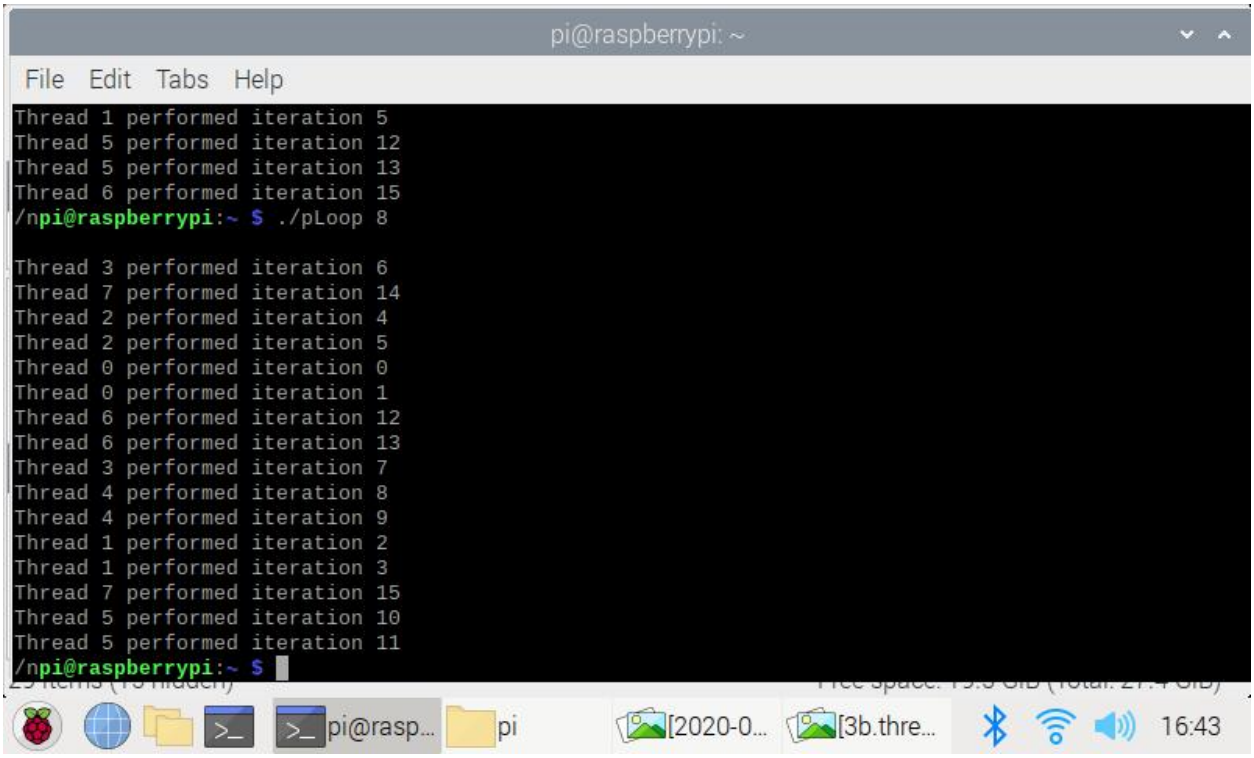


4 threads created and observed that 0 id performed iteration 0, 1, 2, and 3. Id 1 performed iteration 4, 5, 6, and 7, id 2 performed iteration 8, 9, 10, and 11. Similarly id 3 performed iteration 12, 13, 14, and 15. This shows that each thread id is involved 4 in iterating exactly 4

iterations. Also noticed that lower id number is used for initial iterations. And higher id numbers for the later iterations.
Also, the pattern of thread used per iteration has as given has been verified through code..



Tried several other numbers of threads to see a pattern of iteration assignments and noted that initial iterations are performed by lower threads and later ones by higher thread ids.

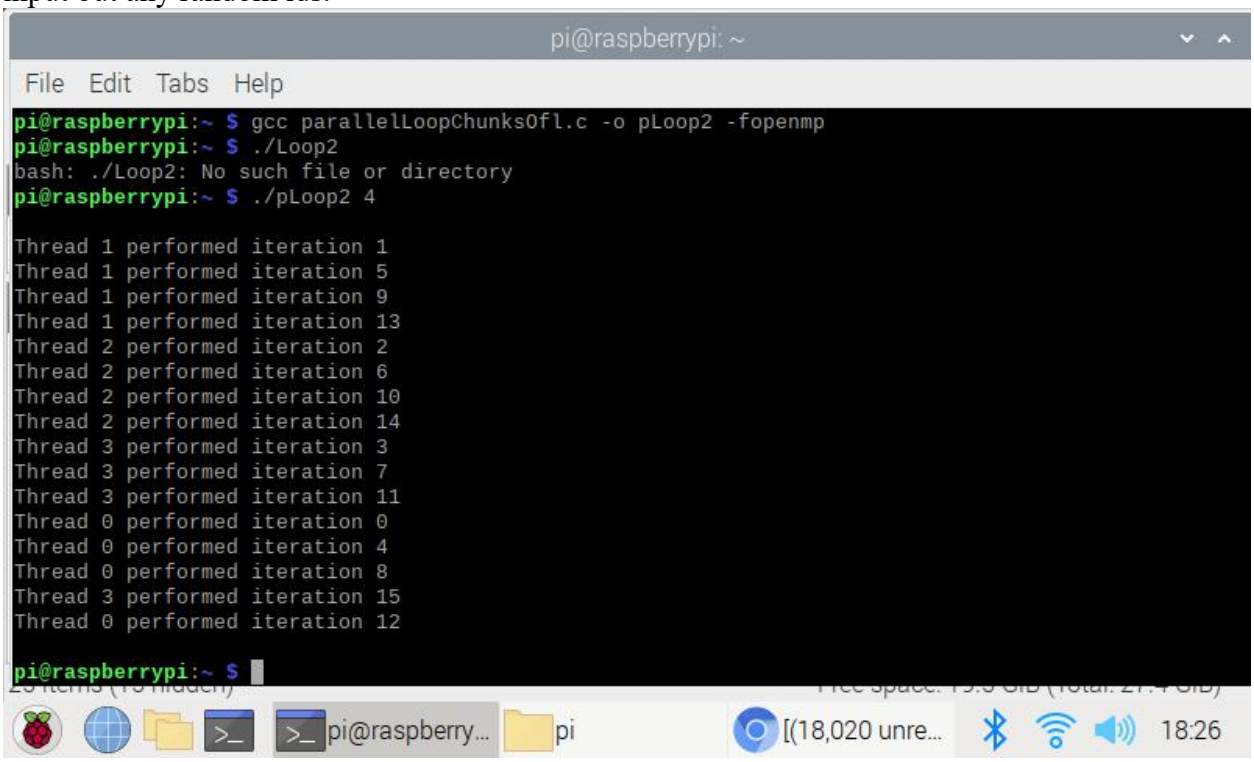


When the number of iterations is not evenly divisible by the number of threads, the even pattern of thread created is no longer possible which is shown in the above screenshots.

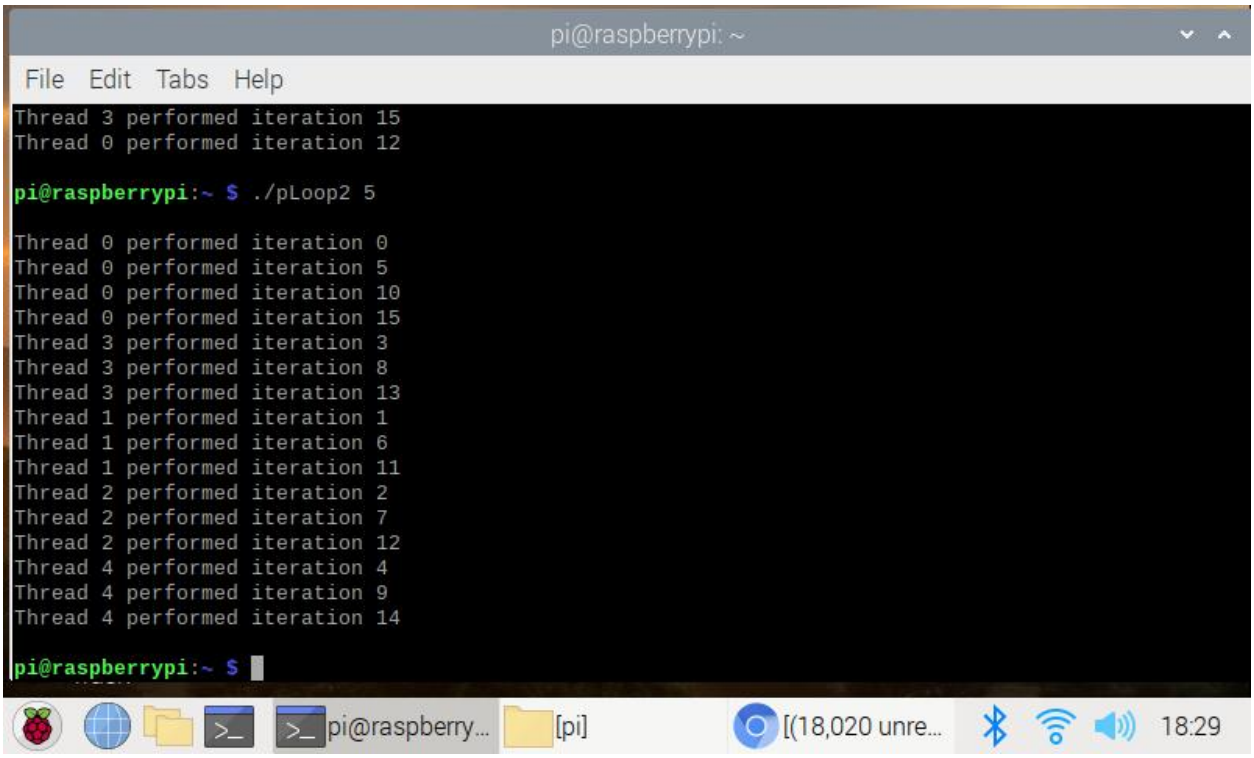
Thread Scheduling

In the first part of the code, we have used static scheduling system in which we set pragma to set schedule statically. We compared the thread performed iteration with the previous version and found no difference compared with this version except a different approach of thread implementation through scheduling.

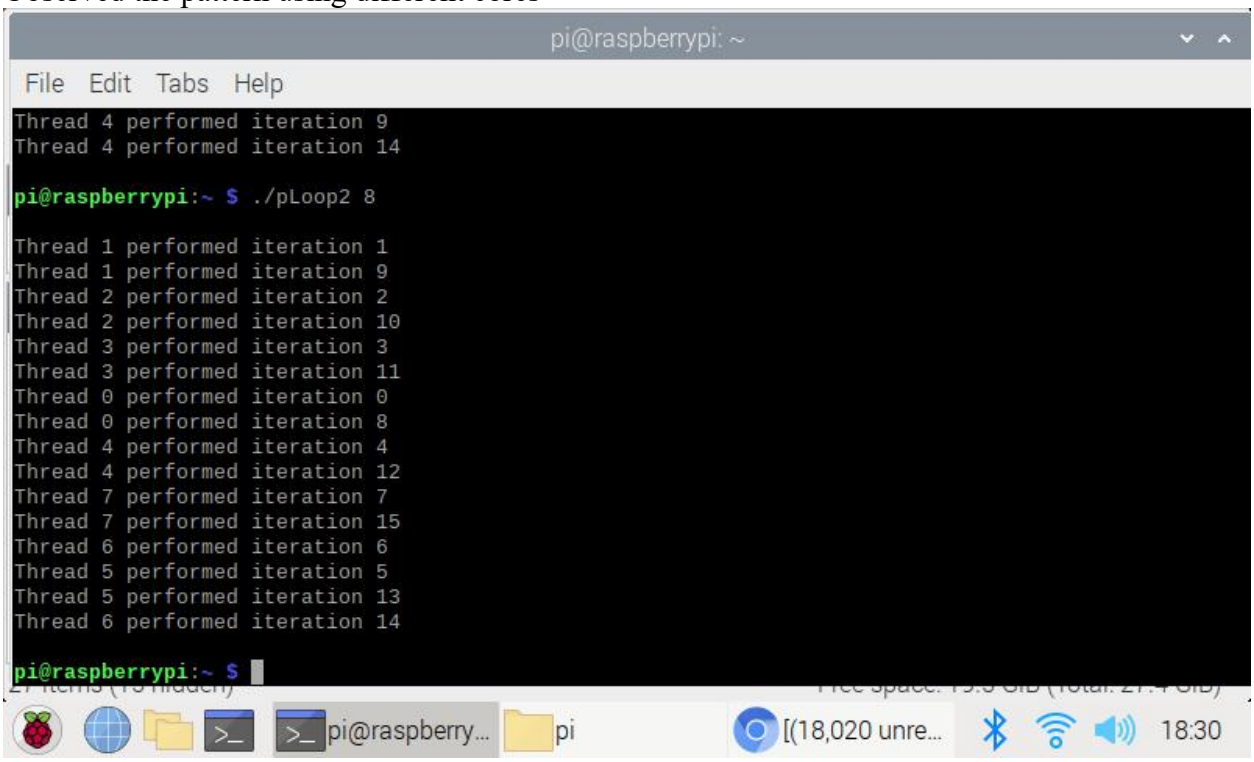
Next, we uncomment the section of code provided and study the iterations performed by the threads. We then use different cores to create threads and observed the iterations performed. We also noted that this time, the static version of the code is taken off from the pragma. Here we noticed that the pattern of output is the same, however, the output of iteration id is not what we input but any random ids.



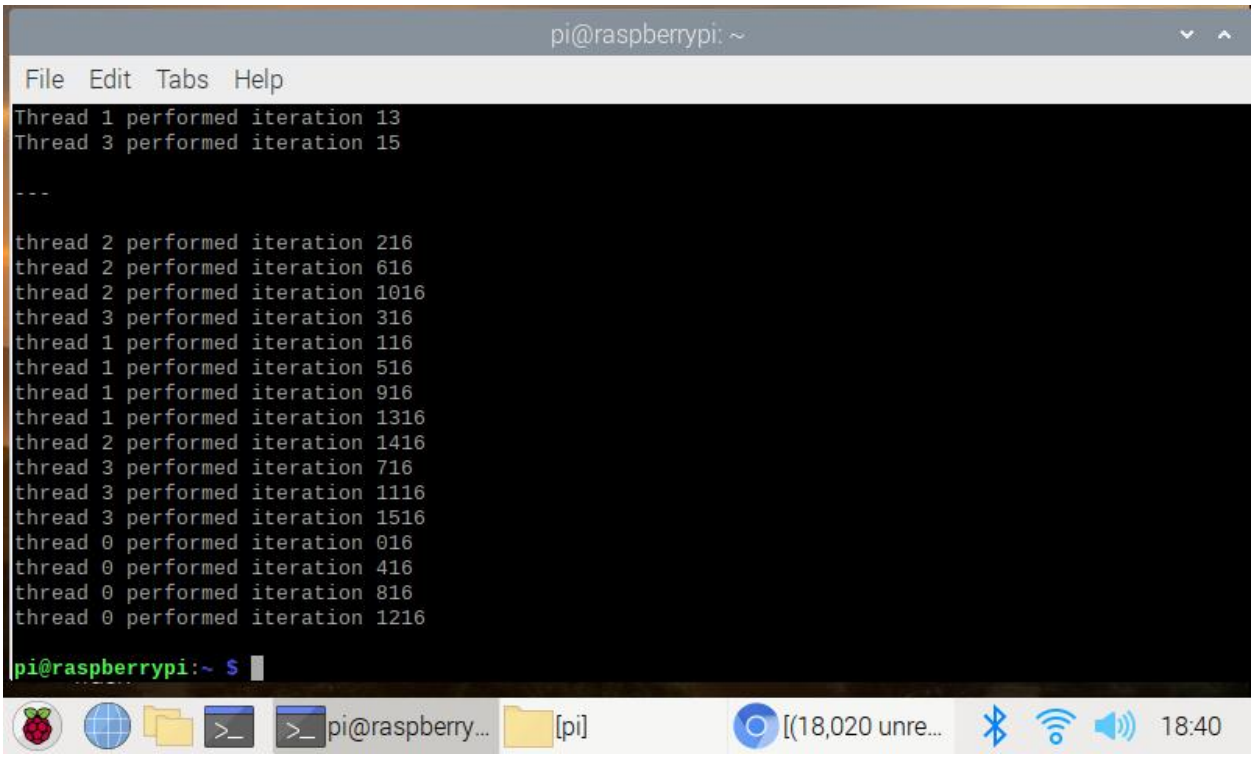
The pattern of thread creation and distribution of iteration formation is quite different to that without using static version.



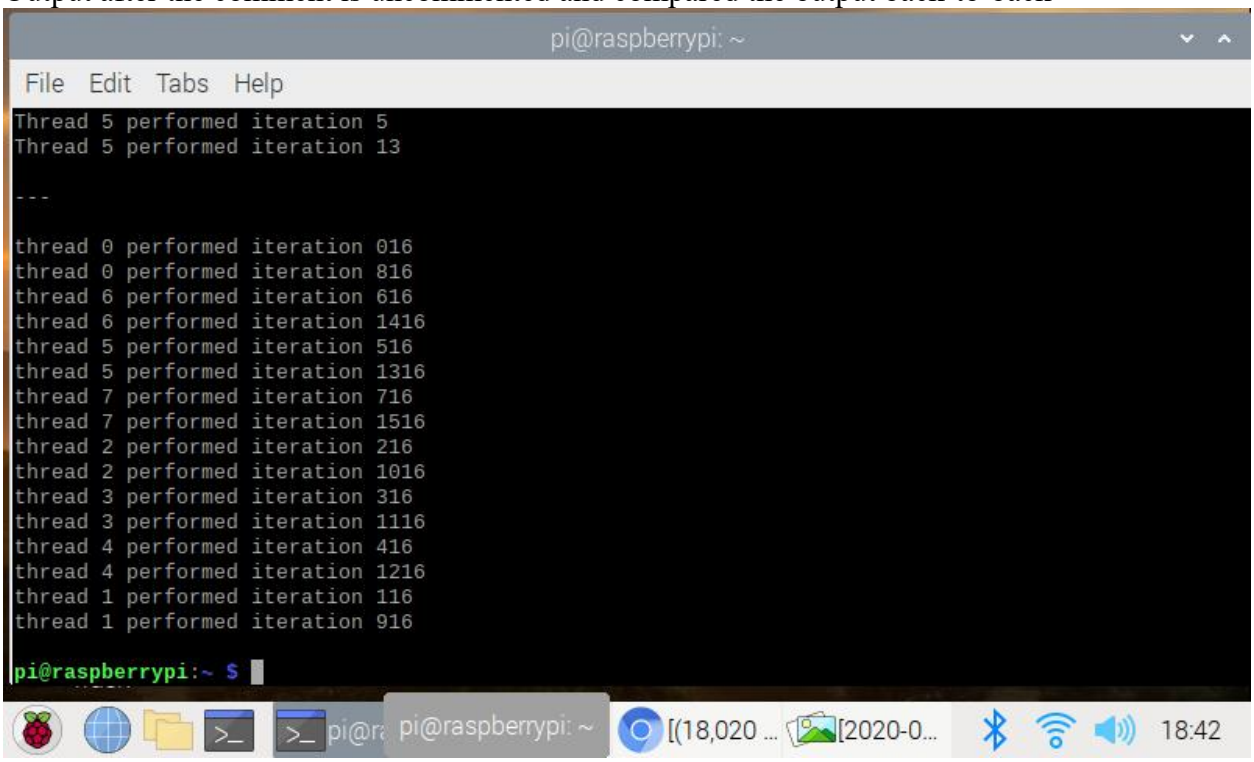
Observed the pattern using different cores



Observed using even multiple of 4



Output after the comment is uncommented and compared the output back-to-back

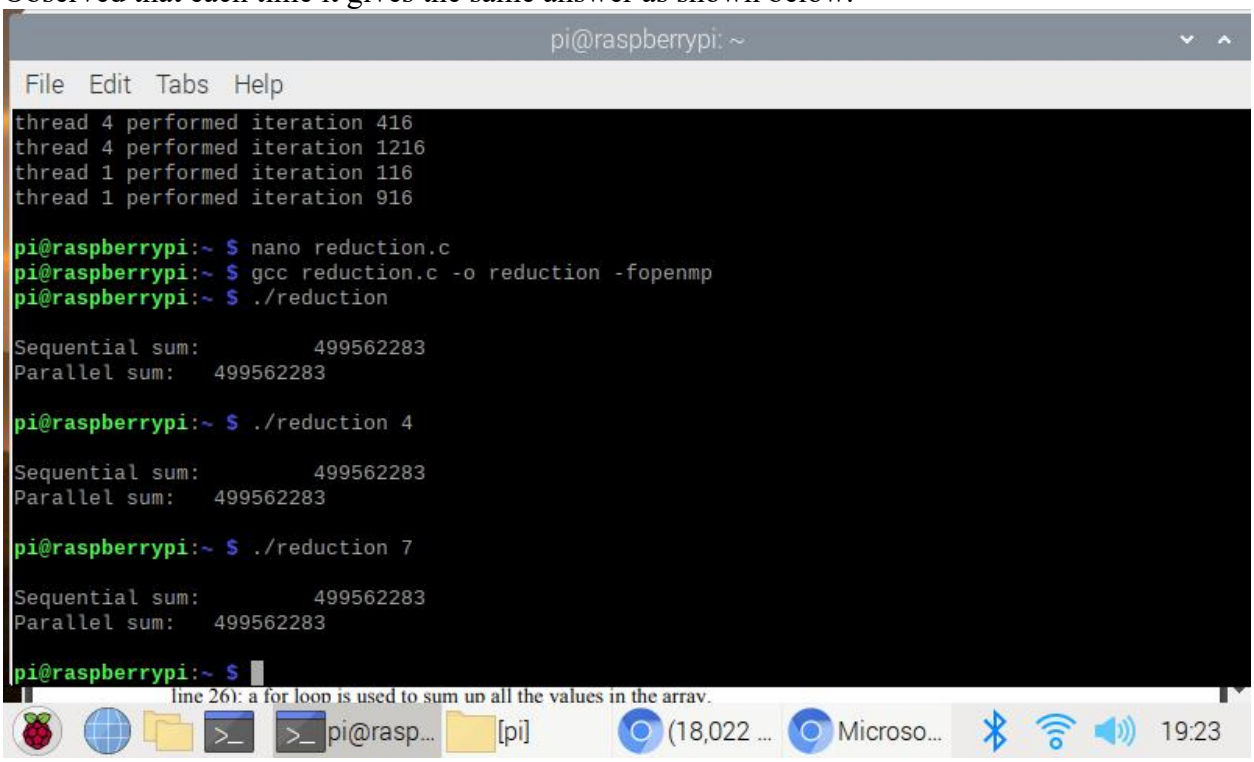


The output is seen the same except the iteration id.

When loops has dependencies

3.4 Dynamic scheduling

Compiled and ran the code as is (without uncommenting the comment) and looked into the output using different number of cores.
Observed that each time it gives the same answer as shown below.



The output is the same regardless of number of threads created. This is because the parallel computation has not taken into effect, yet.

Show and explain the output after running the program.

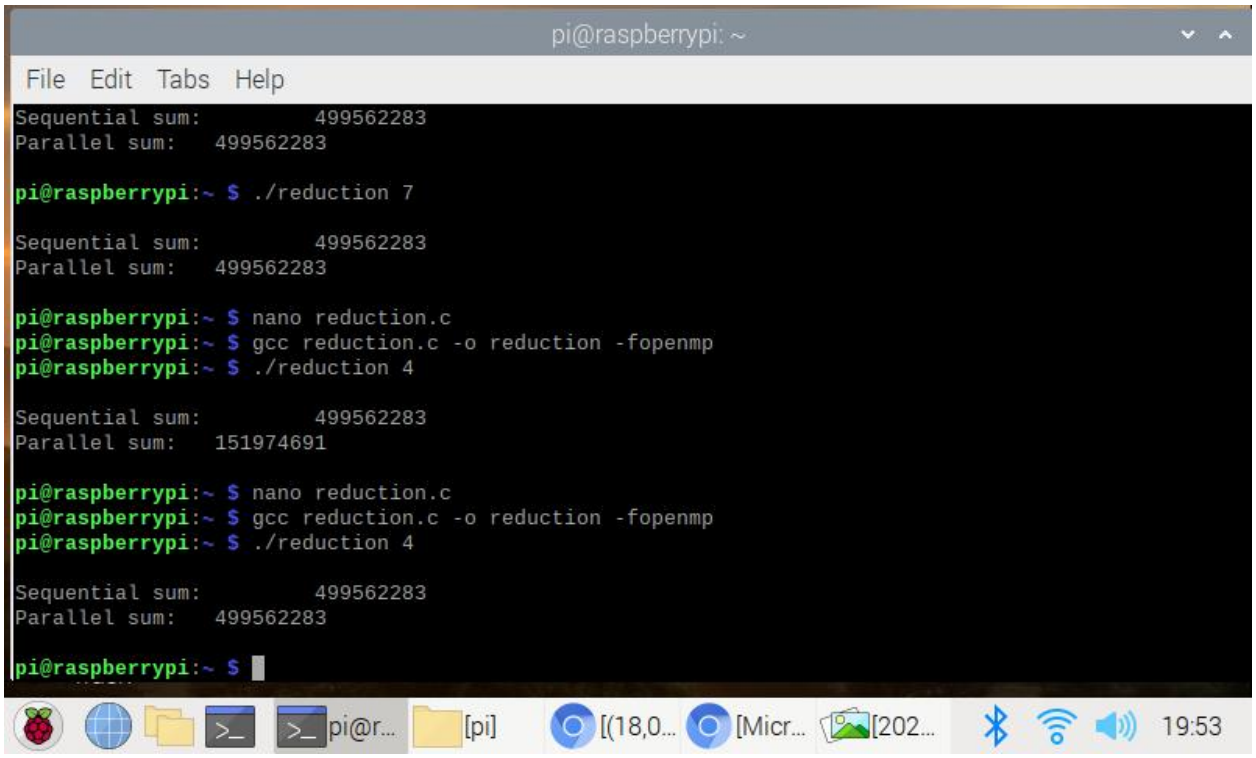
The sequential and parallel sum came up to be the same. However, we have not used the parallel computation yet as we have commented the pragma

1) To actually run it in parallel, per instruction, we uncommented the first comment and investigated the output.

In this case we get different answers for the:
sequential sum (49562283), and
parallel sum (151974691)

2) This time we removed the second comment and let the parallel accumulate the total sum coming from all cores.

This time the answer on both the summation methods matched as shown in the screenshot



Code uncommented and investigated and each time output is studied.

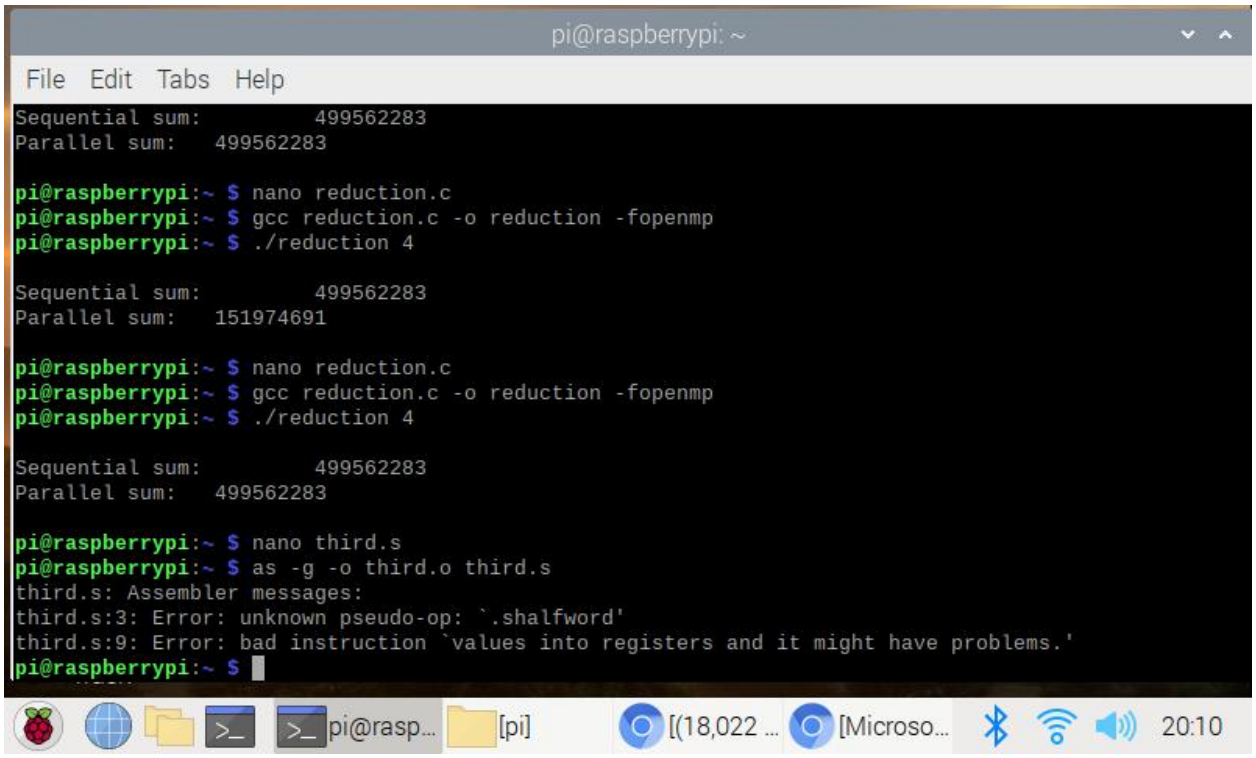
TASK 4: Arm assembly programming.

<u>TASK 4a</u>

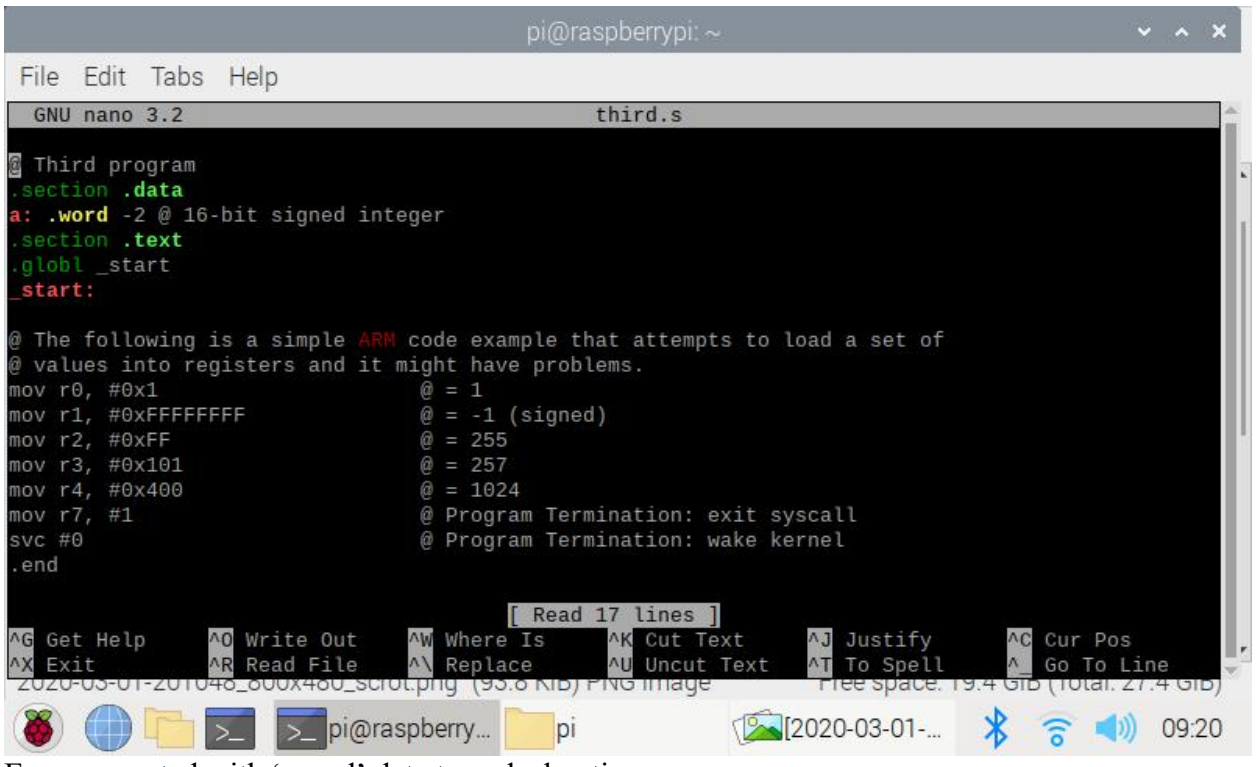
Edited the code into the compiler.
Found that the code has the error while trying to assemble it.

- What error message did you get and why? (Report that)

Answer: Error message: unknown pseudo-op: '.shalfword'
Bad instruction 'values into the registers and it might have problems.'
<u>Why?</u>
.shalfword is not a valid data declaration. It only takes .word as a data declaration.

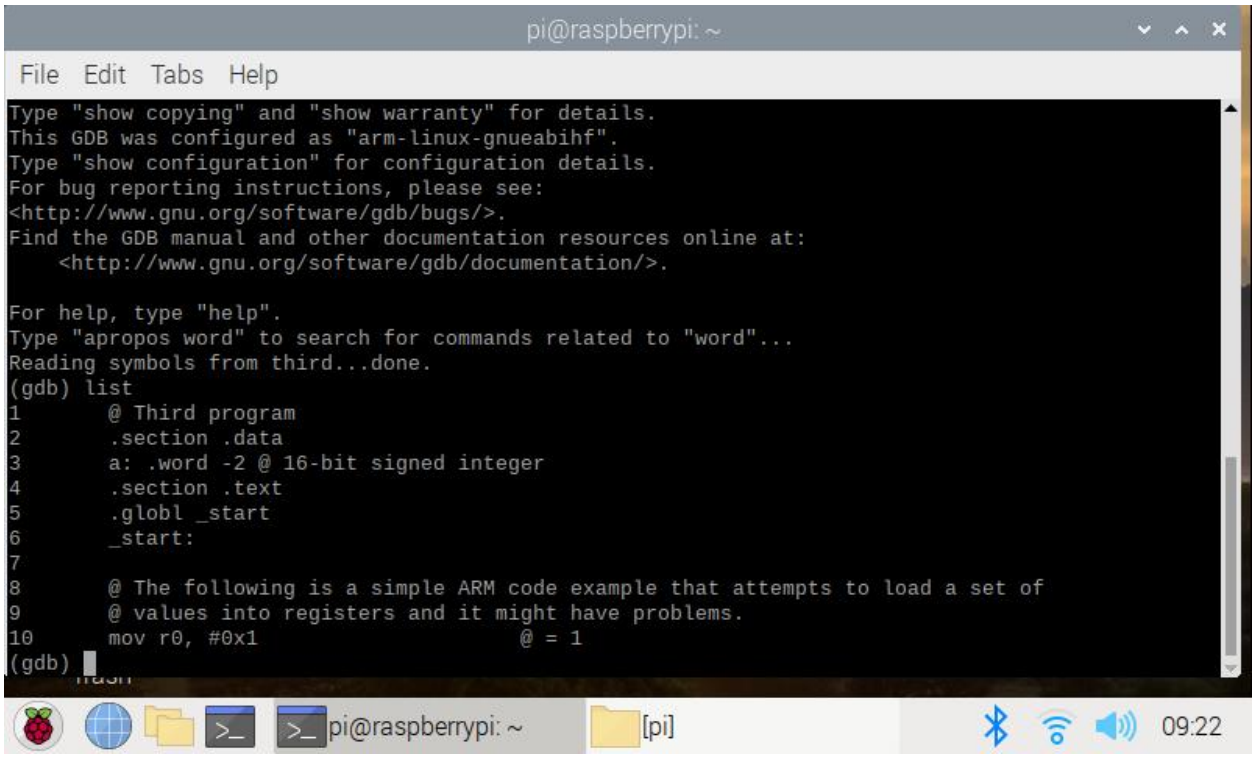


Error noticed.

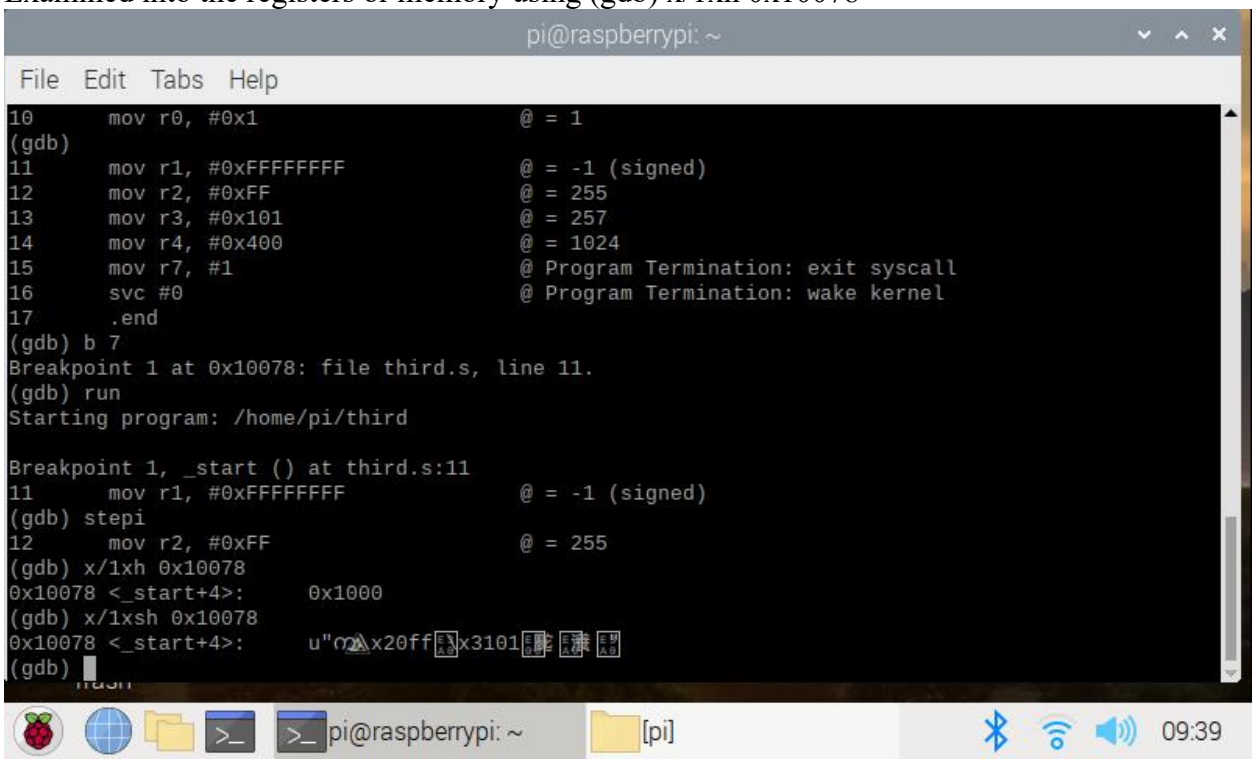


Error corrected with '.word' data type declaration.

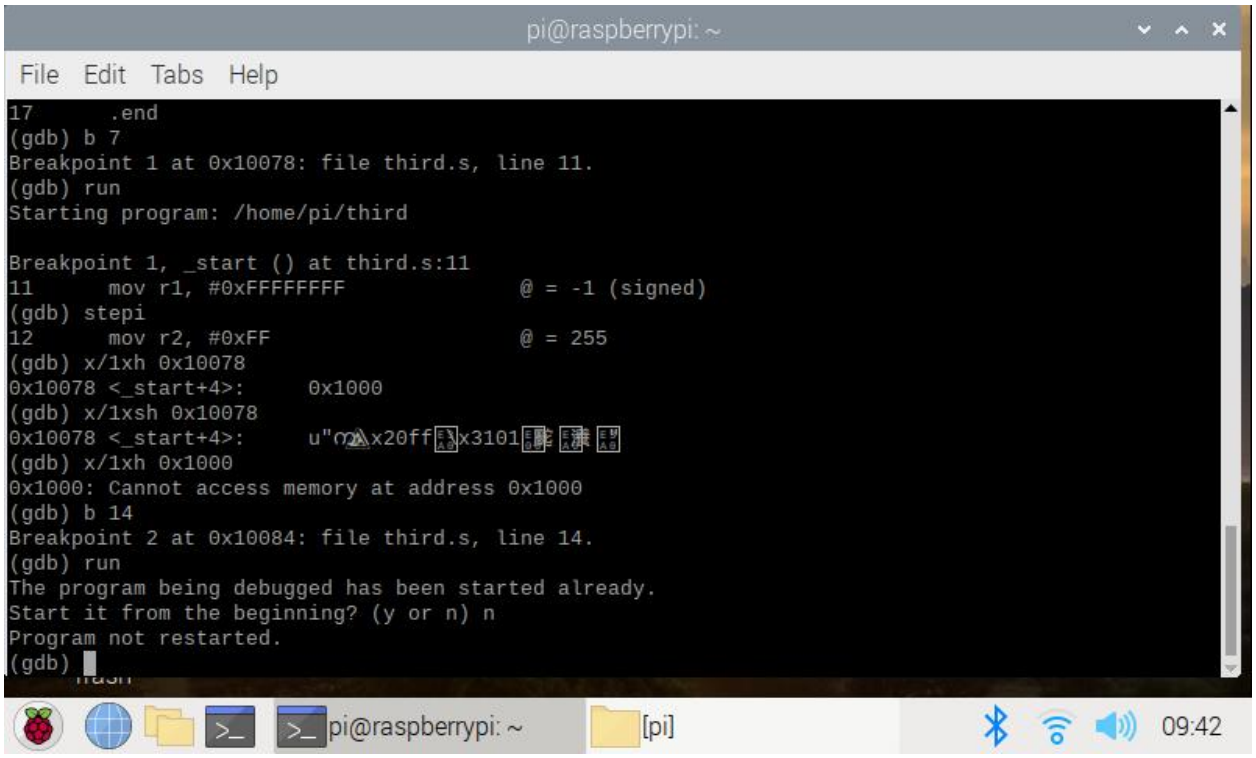
After correction, the code is reassembled, linked, and ran with a debugger as shown below.



Breakpoint created at line 7 but the it was automatically moved to line 10 by the gdb.
Examined into the registers or memory using (gdb) x/1xh 0x10078



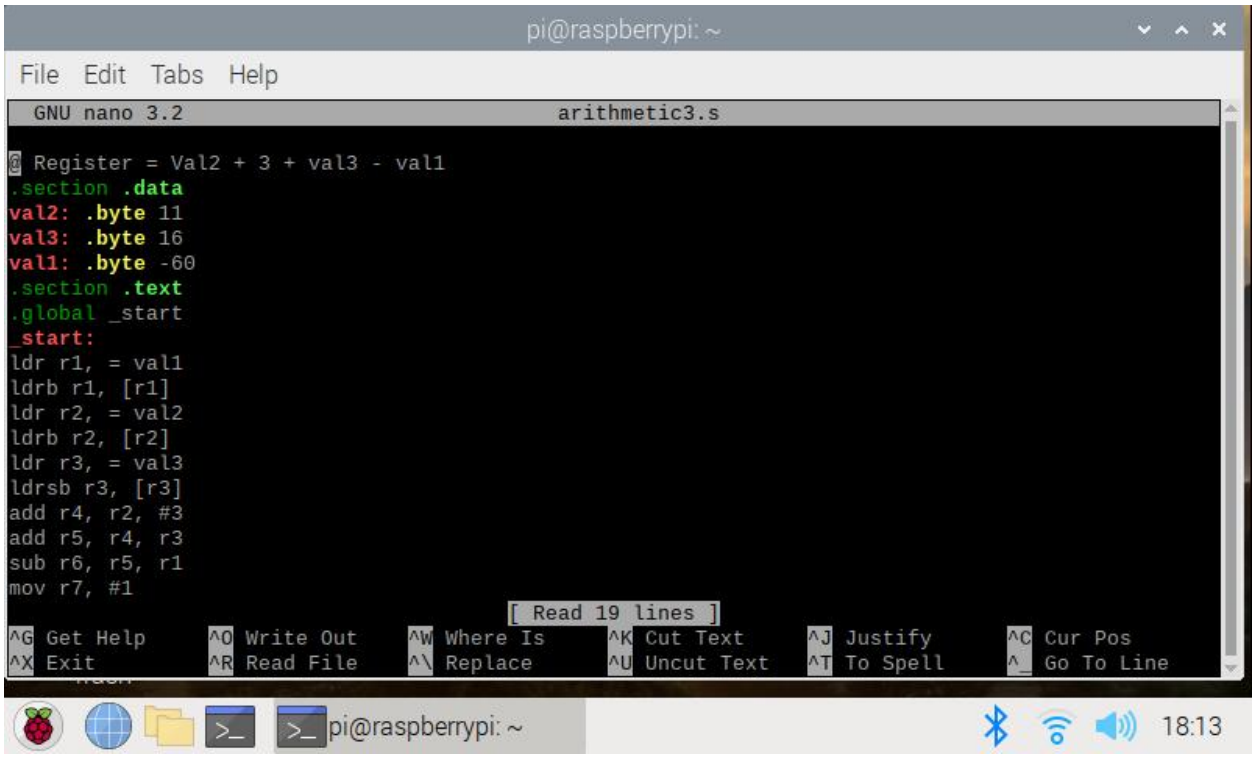
With sh(signed), instead of h(unsigned) got into the same upset, however, an unknown address is generated that I am unable to understand. screenshots of observation is shown above and below.



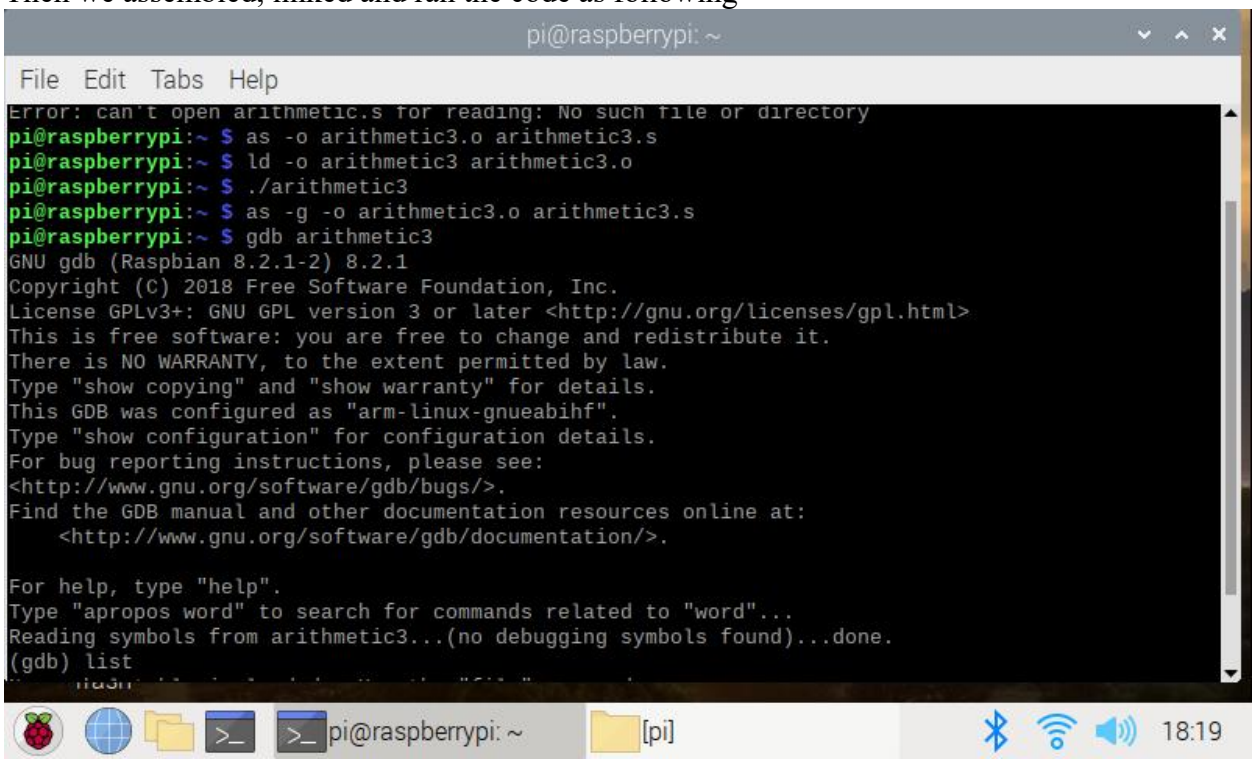
Tried sh instead of h and observed the changes inside the registers/memory. Also tried using different offset address but couldn't understand what's going on.

Task 4b

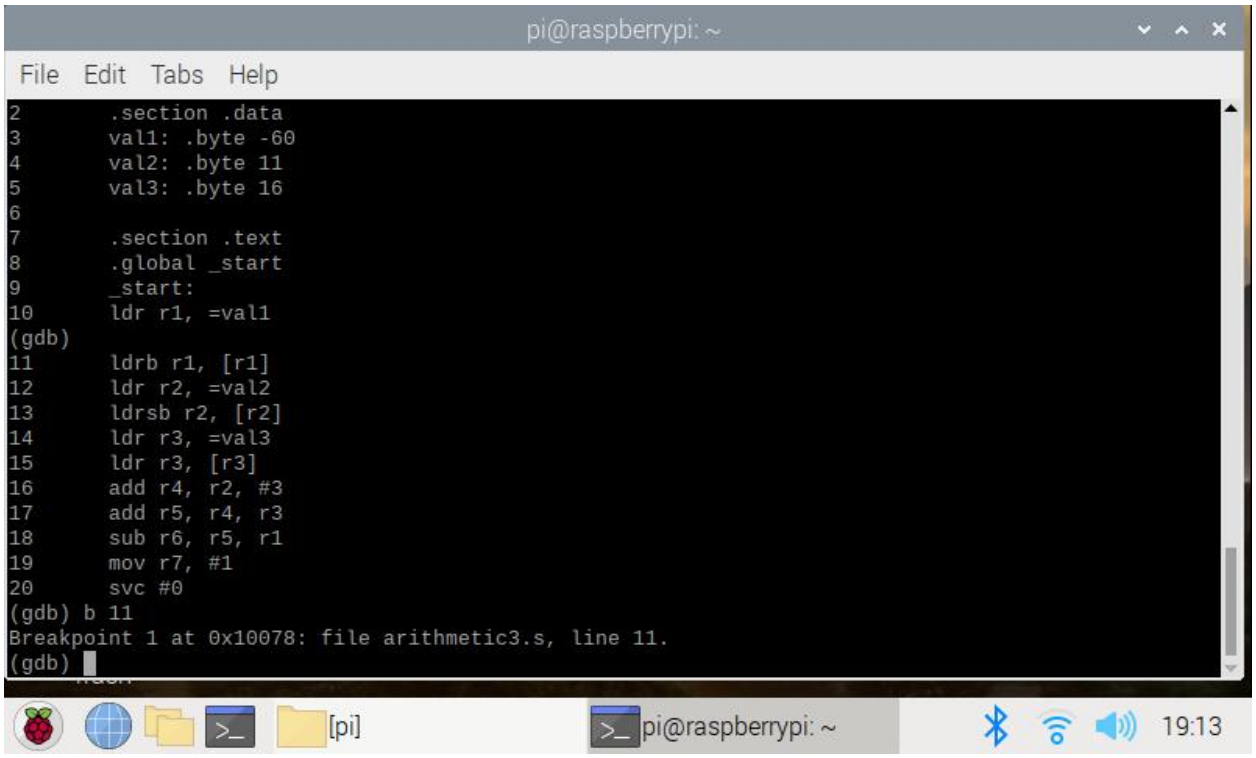
Referring to third.s as a base, and with the algorithm
Register = val2 + 3 + val3 – val1; val1 = -60(signed), val2 = 11(unsigned), and val3 = 16(unsigned) and all these memories are 8-bit integer memory
we worked together and created the following ARM assembly code with the name arthmetic3.s



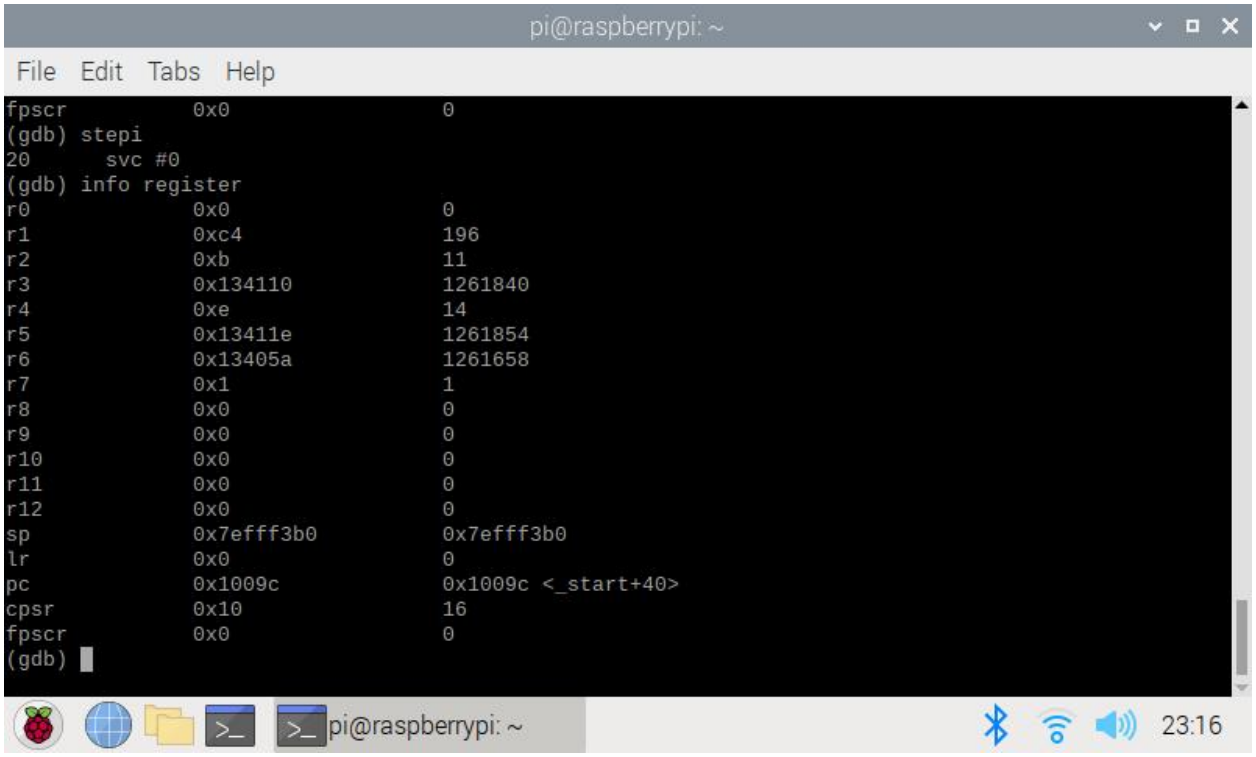
The code compiled
Then we assembled, linked and ran the code as following



The code assembled, linked, debugged, created a breakpoint at line 10, and ran

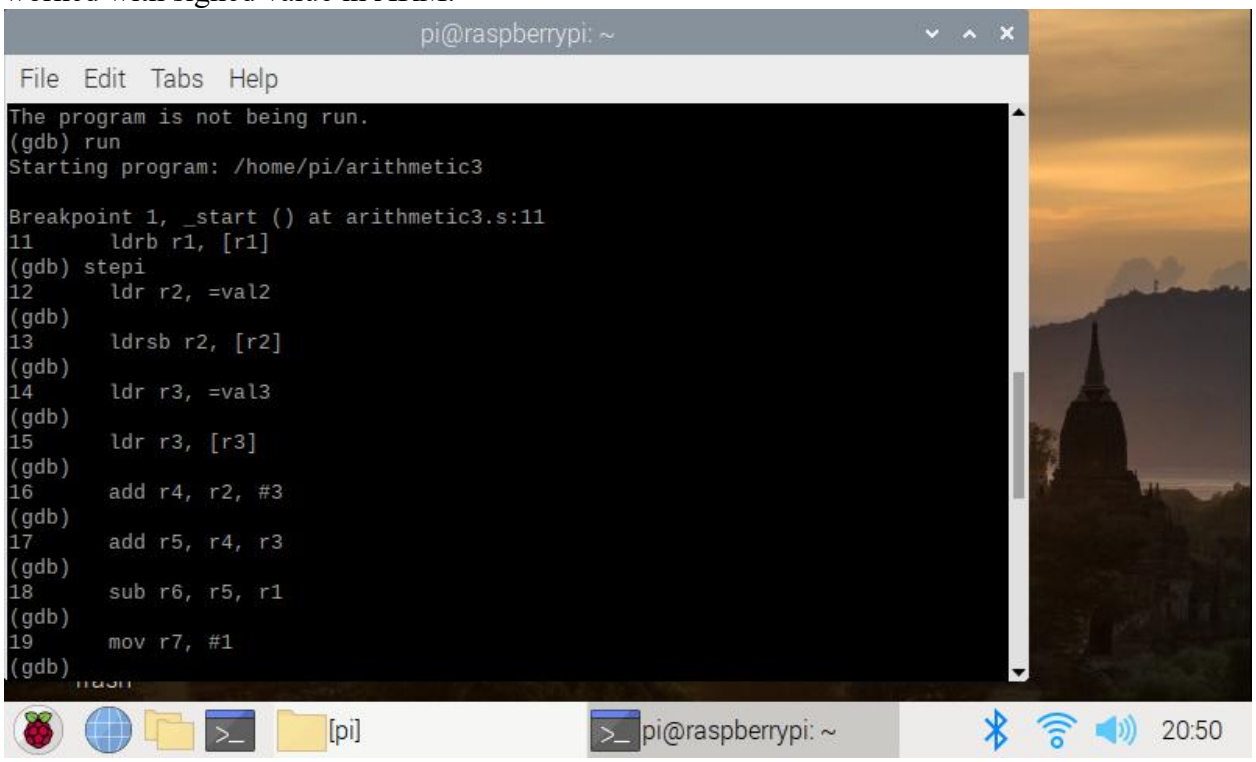


Debugged with breakpoint at 10



The assigned arithmetic to be computed with their assigned value is: 11+3+16-(-60) = 90.
Our

r1 = C4h which corresponds to -60 (11000100) that is our Val1
r2 = 0Bh which corresponds to 11 that is our val2
r3 = 10h which corresponds to 16 that is our val3
similarly
r4 = 14
r5 = 30
r6 = 90 (5a) which is our final answer of the "Register"
all these final and intermediate arithmetic values are as expected.
Upon checking the 'cpsr' value is 0x10 which tells us that the negative flag is set as we worked with signed value in ARM.



We went further to learn more with the stepi and check some details of the values in memory and registers of our program.
Stepped in into each register and studied the value of registers

Vedansi Parsana

TASK 3: Parallel programming skills

Foundation:

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

→A task is an instruction set or a program that is executed by a processor.

→Pipelining is a type of parallel computing where task is segmented into individual steps which can then be completed by different (multiple) processor units.

→Shared Memory, according to computer architecture system, multiple processors share direct access to a common physical memory mostly through the bus. According to programming, parallel tasks are said to have same "picture" of memory and can access any of the memory, it can also be accessed directly. Even if we don't know where the physical copy is saved.

→Communications is the process of exchanging data when computing parallel tasks, There are many other ways such as using shared memory bus or network but the best way to exchange data while computing parallel tasks is through Communication.

→Synchronization is a branch of communications focused on the coordination of parallel tasks. It halts a task until other tasks haven't reached the same point which prevents the supported tasks from trying to execute before they are able to be compute.

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

Flynn's Taxonomy is a common way to classify parallel computers. It is a system which categorizes multi-processor computer architectures into one of four categories which are:

- SISD (Single Instruction, Single Data) is the model in which, during any one clock cycle, only one sequence of instructions is being acted on the CPU and only one data stream is being used as input. It is an oldest type of the computer.

- SIMD (Single Instruction, Multiple Data) is the model in which, during any clock cycle, all the processing units can execute same set of instructions and every unit can work on different data element. Mostly modern processing computers use SIMD.

- MISD (Multiple Instruction, Single Data) is the model in which, during any one clock cycle, each processing unit can operate independently through different instructions though only one stream is being used as input among multiple processing unit. There are not many computers existed of this stream.

- MIMD (Multiple Instruction, Multiple Data) is the model in which, during any one clock cycle, each processing unit can execute a different set of instructions while each processing unit can utilize a different data as input. Modern and supercomputers are MIMD.

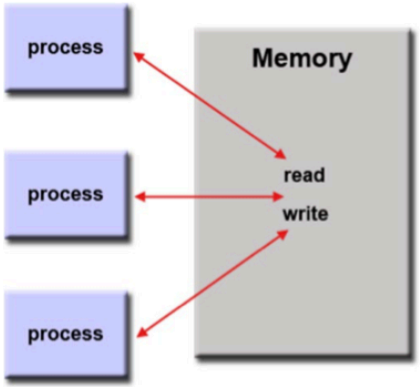
What are the Parallel Programming Models?

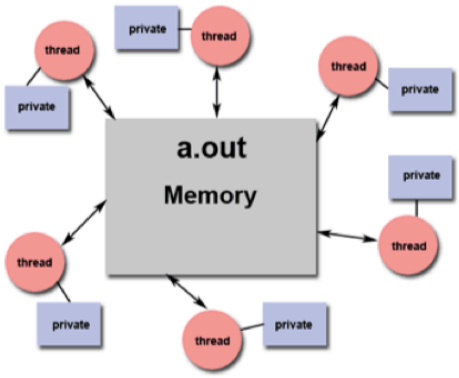
A parallel programming model is a set of abstractions which are upon the hardware and memory architectures of the given system. The commonly used Parallel Programming Models are: 1) Shared Memory (without threads), 2) Threads, Distributed Memory / Message Passing, 3) Data Parallel, 4) Hybrid, 5) Single Program Multiple Data (SPMD), 6) Multiple Program Multiple Data (MPMD).

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

There are two general types of Parallel Computer Memory Architectures: shared memory architecture and distributed memory architecture. Shared memory architecture allows all processors equal access to the same memory resources as global address space. The memory access time given to each processor are of two types: Uniform Memory Access machines and Non-Uniform Memory Access Machines where processors have equal access time and where access time can differ between processors respectively. Distributed memory architecture operates as a network of many local memories which belongs to every single processor. OpenMP utilizes a shared memory architecture and it must utilize shared memory architecture because it allows the application to utilize a multithreading model. Since threads must share a common memory with the process to which they belong, and each thread is assigned parallelly to an individual processor.

Compare Shared Memory Model with Threads Model?
Shared Model → Left. Thread Model → Right.
The Shared Memory Model utilizes a single shared address space for all. It utilizes many controls in order to regulate access to the shared memory and resolve any issues that could arise from competition for the shared memory resource. The Threads Model focusses on a single main process (a.out) that has multiple threads which share a memory space. Each thread has their own private, local data but also have equal, concurrent access to a shared memory which is a.out memory. This access is/can be managed by synchronization.





What is Parallel Programming?

Parallel programming is the utilization of multiple processors in computation. While serial programs execute tasks one after the other using a single processor, parallel programs execute tasks simultaneously over multiple processors which significantly increases the output of the system.

What is System on a Chip (SoC)? Does Raspberry PI use system on SoC?
System on chip is a modern approach of reducing the size of a computer by integrating CPU, RAM, GPU on a single microchip instead of using different chips. The Raspberry PI utilizes a SoC unit.

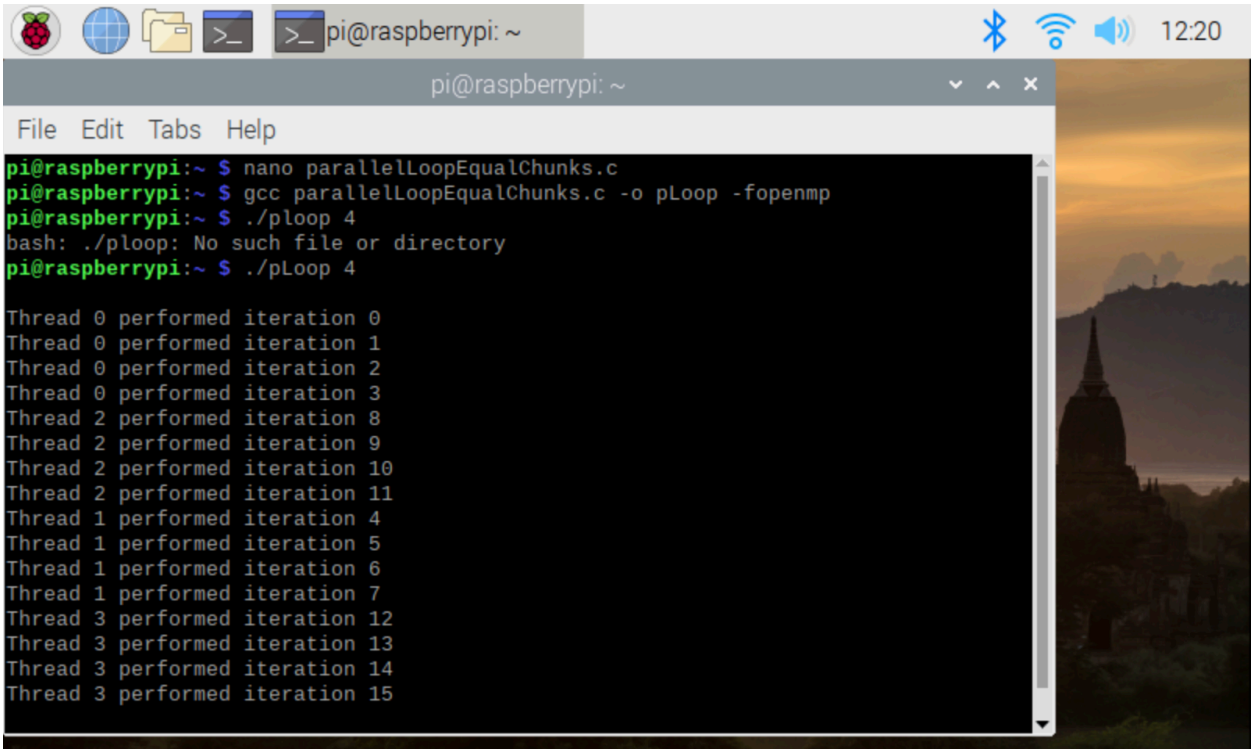
Explain what the advantages are of having a System on a Chip rather than separate CPU, GPU, and RAM components.

A SoC is about the size of a CPU while containing all of the necessary components of a computer system, making it possible to have smart phones and even watches. Due to being so much smaller and being self-contained. It costs less to build than separate CPU, GPU, and RAM components. It also less power than the sum of comparable parts which makes them cheaper.

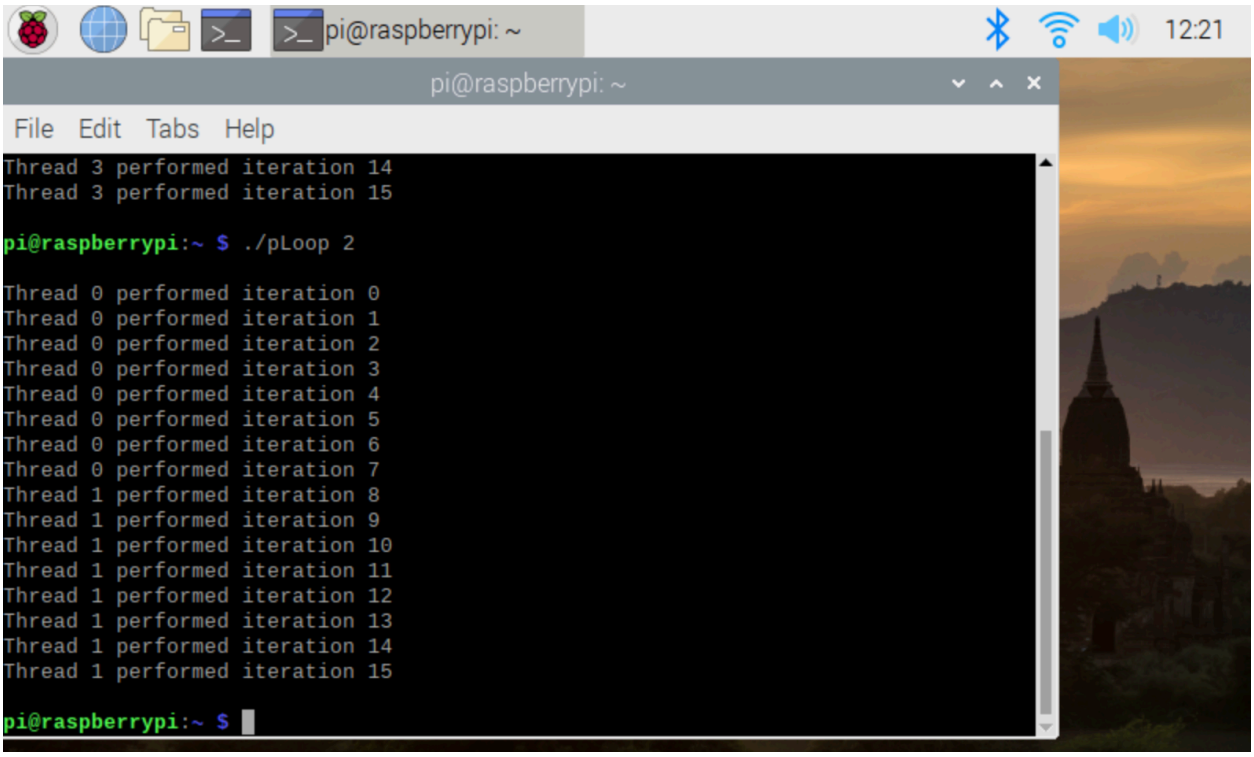
Parallel Programming Basics:

I created a C file named parallelLoopEqualsChunks.c and make it an executable program. With the numbers of iterations divisible by the number of threads, it will divide them evenly.

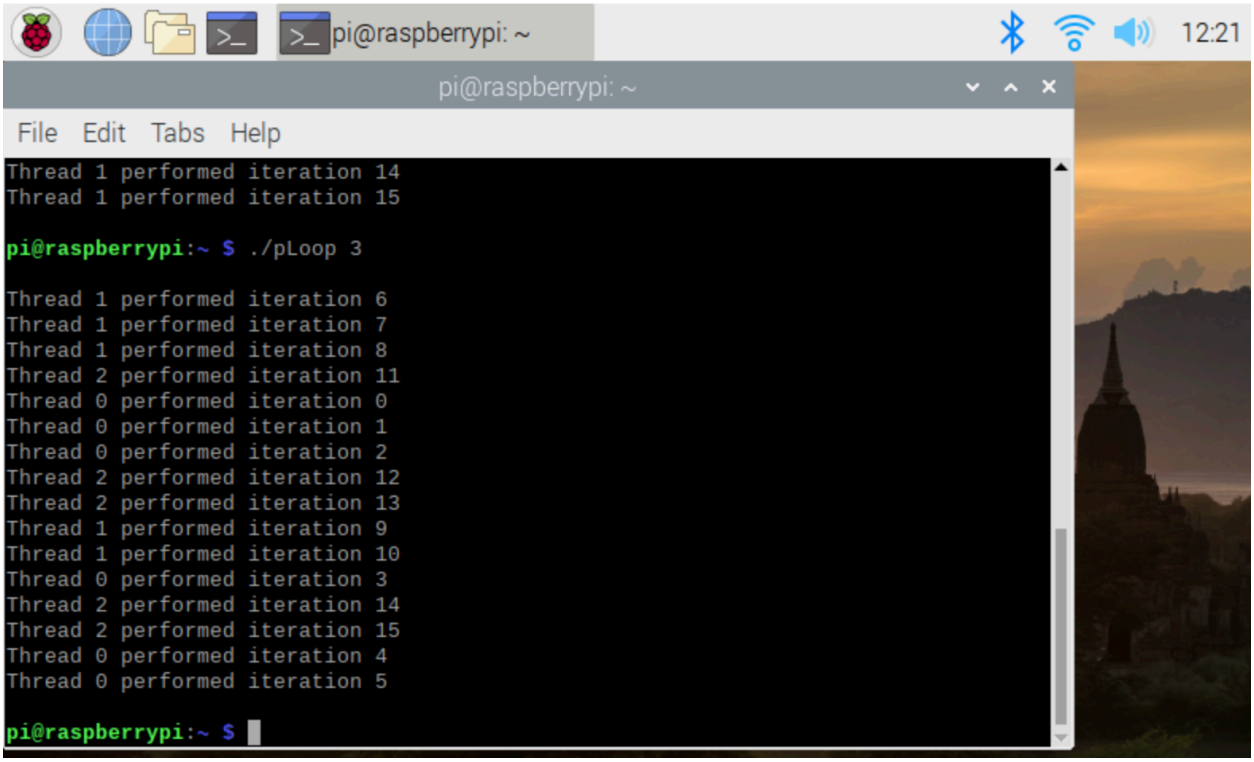
With 4 threads:



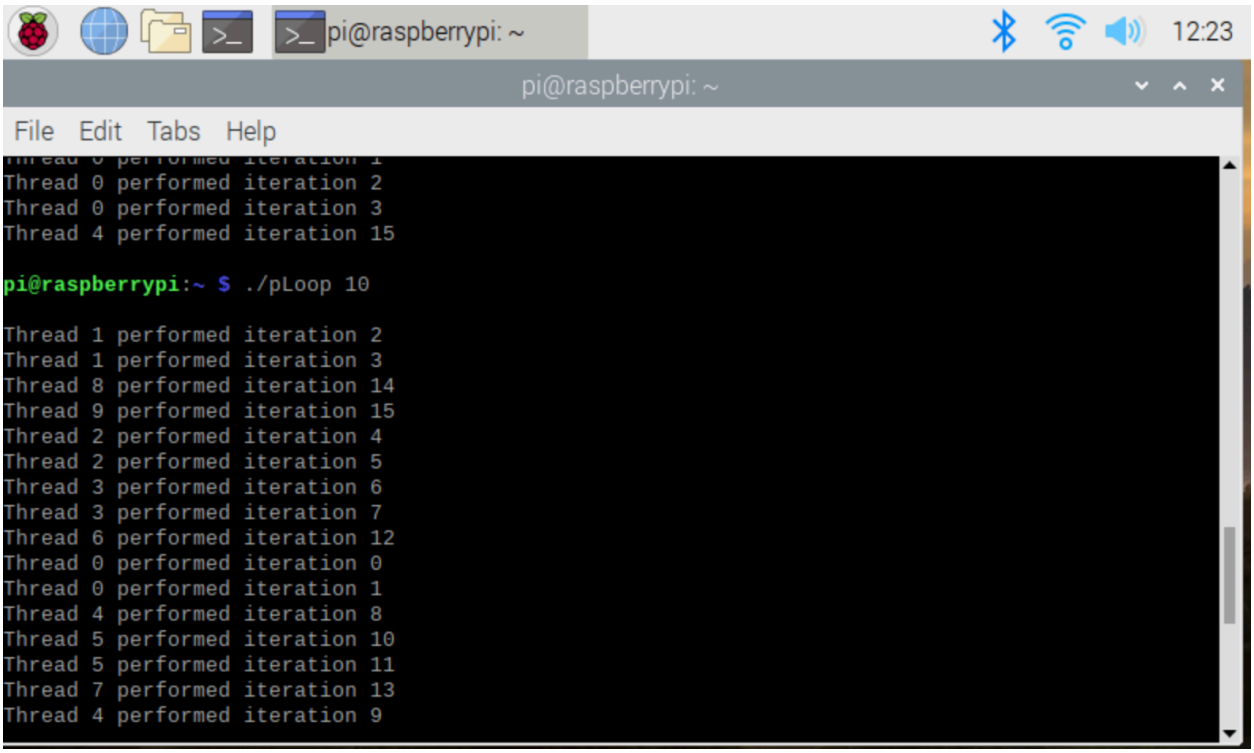
With 2 threads:



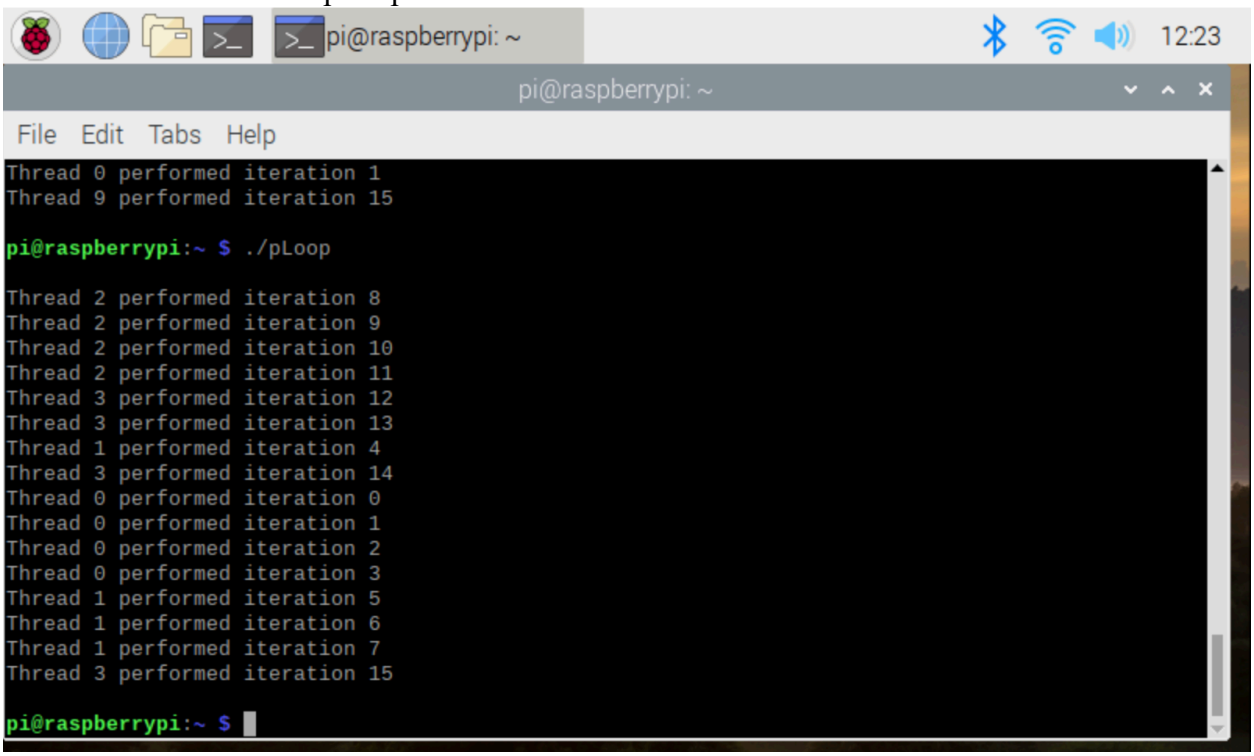
With 3 threads:



With 10 threads:

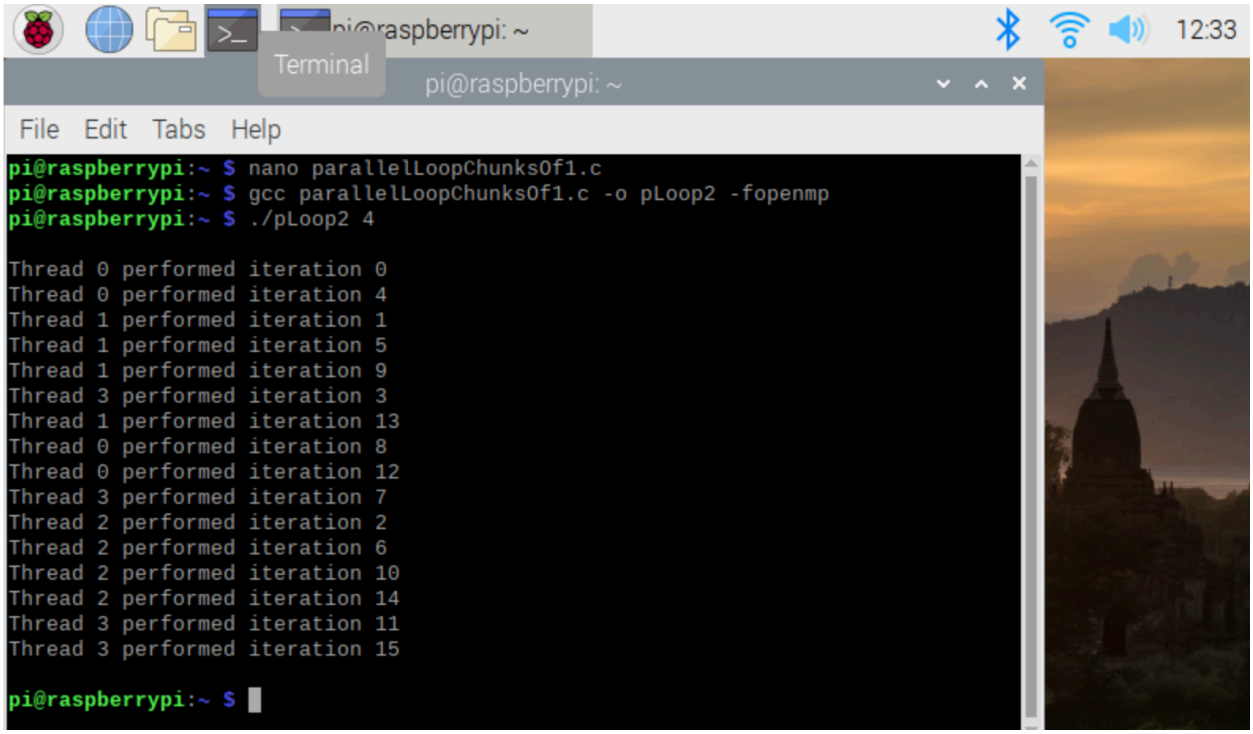


Leave off number after ./pLoop

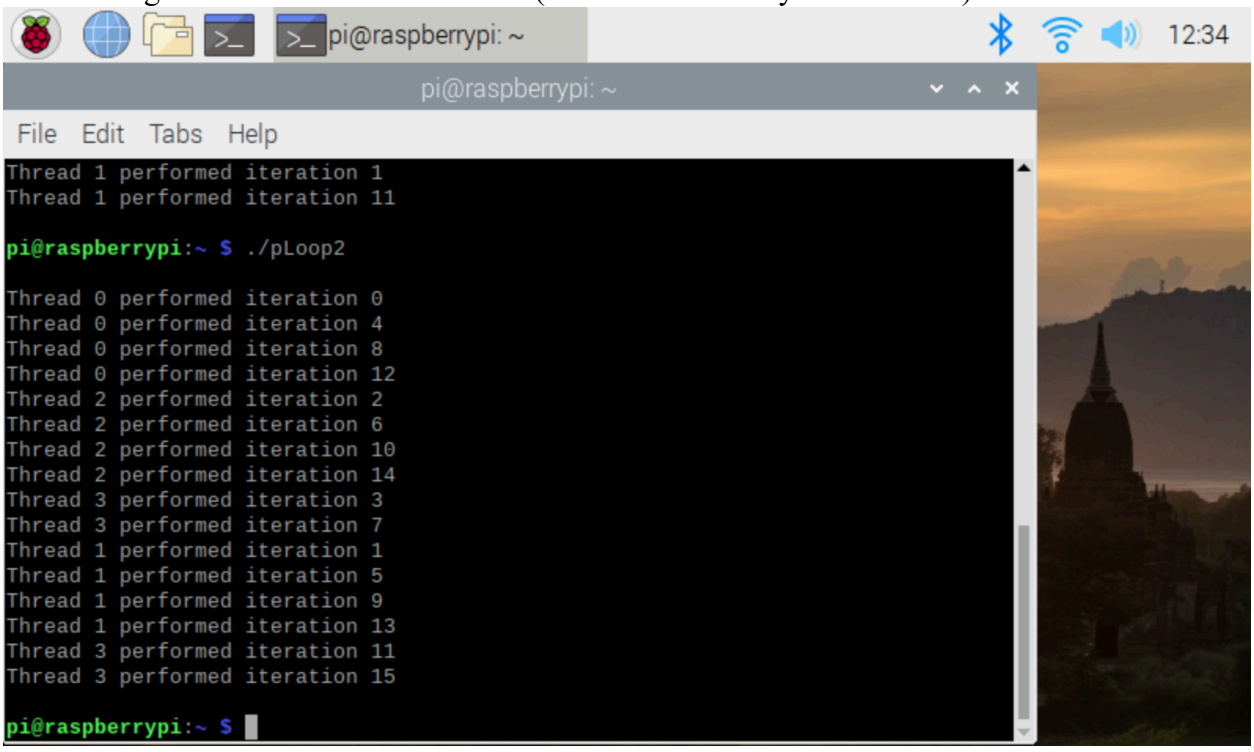


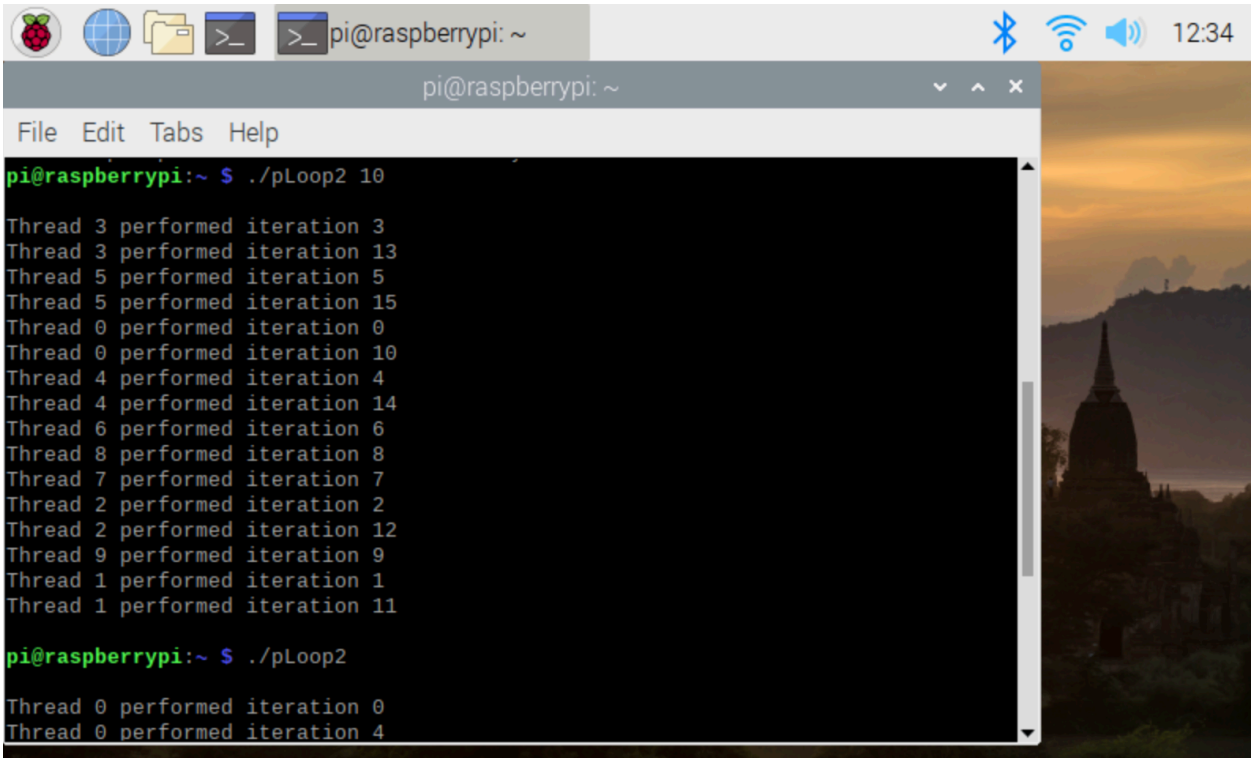
Next, I created a new C file named parallelLoopEqualsChunksOf1 and made it an executable file. In this program we are using (static,1) in the code, which force each of the thread to take only 1 iteration and pass the next iteration for the next thread from thread 0 to thread 3.

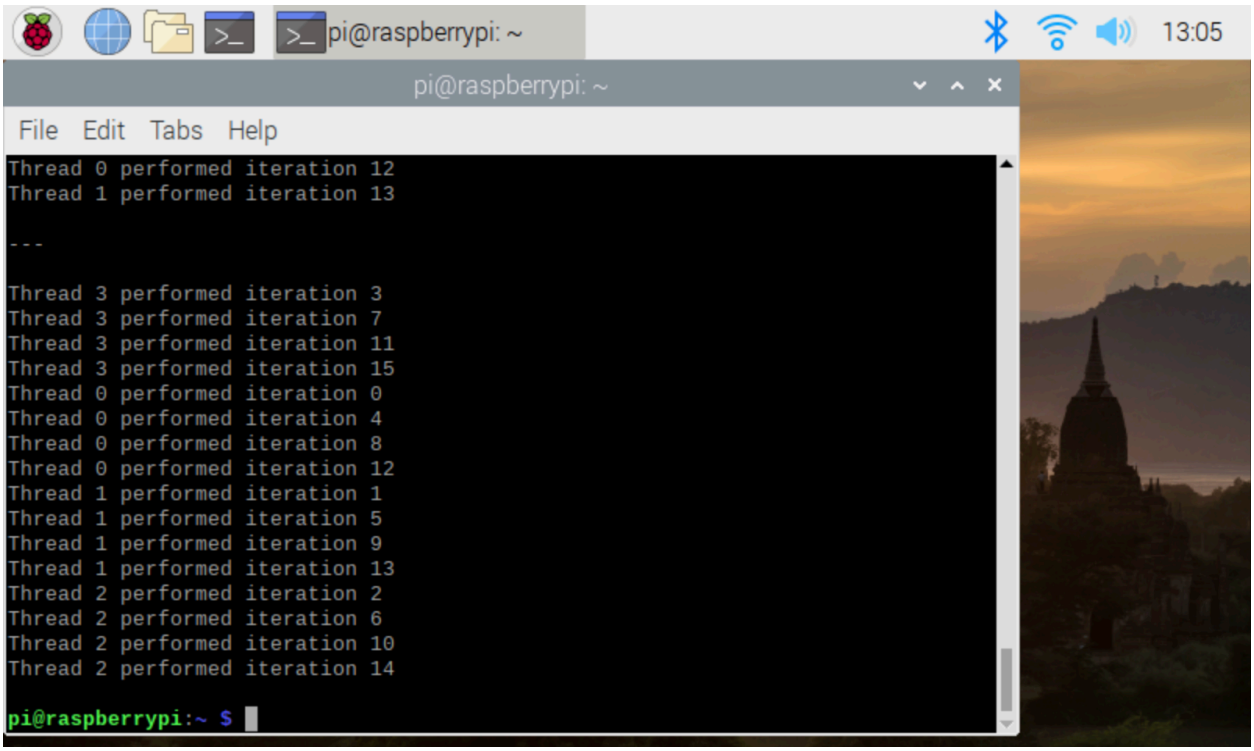
With 4 threads:



The system will automatically assign the number of threads to be 4, since the Raspberry PI has 4 cores and return the results the same with the command ./pLoop2 4. With 10 threads: (./pLoop2 10) It will force only one iteration from thread 0 to thread 9 and then force the remaining iterations again from thread 0 to thread 5 (since there are only 16 iterations).

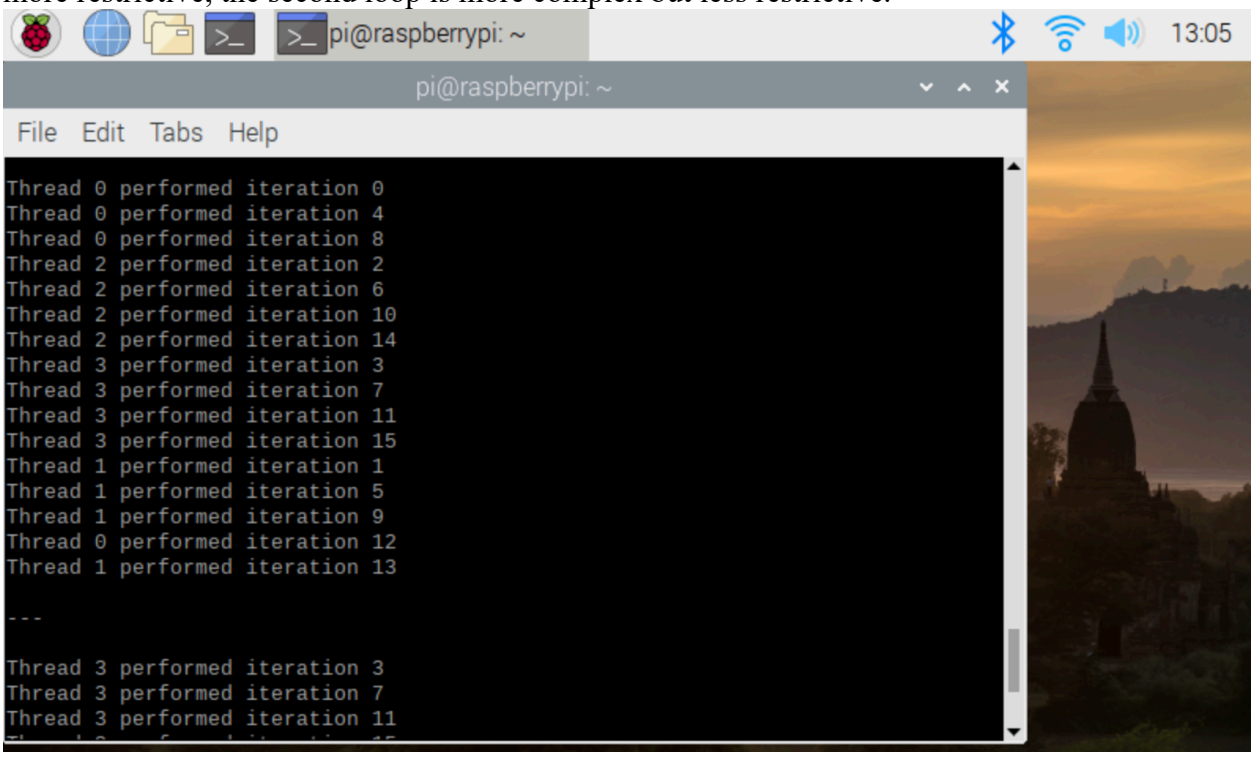




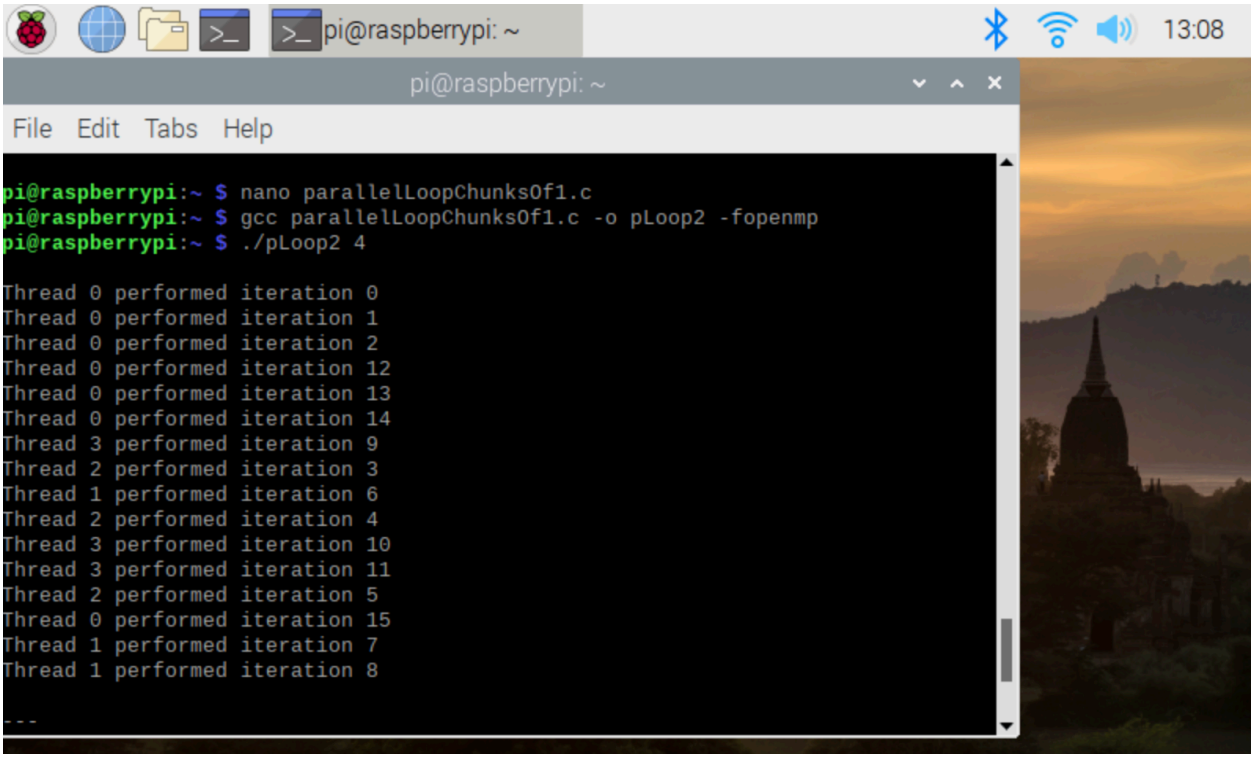


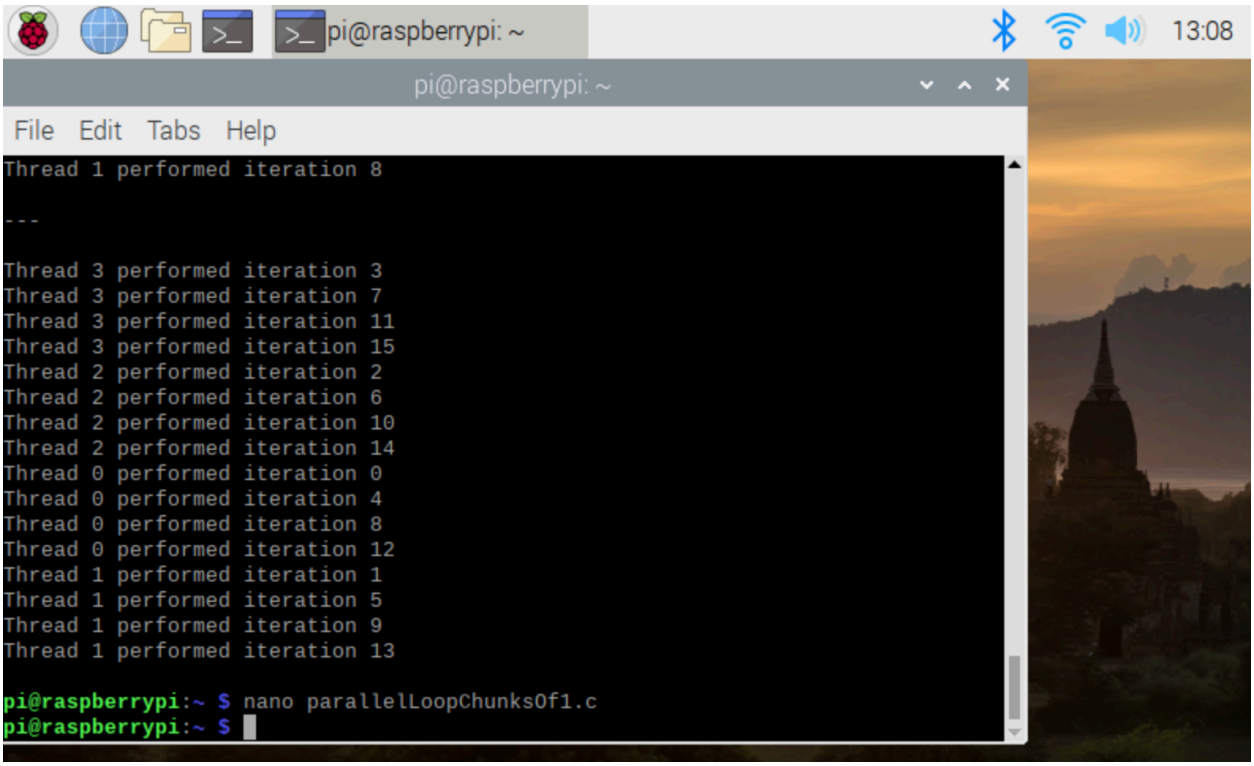
uncomment the second loop, both loops produces the same output. The first loop is simpler but

more restrictive, the second loop is more complex but less restrictive.

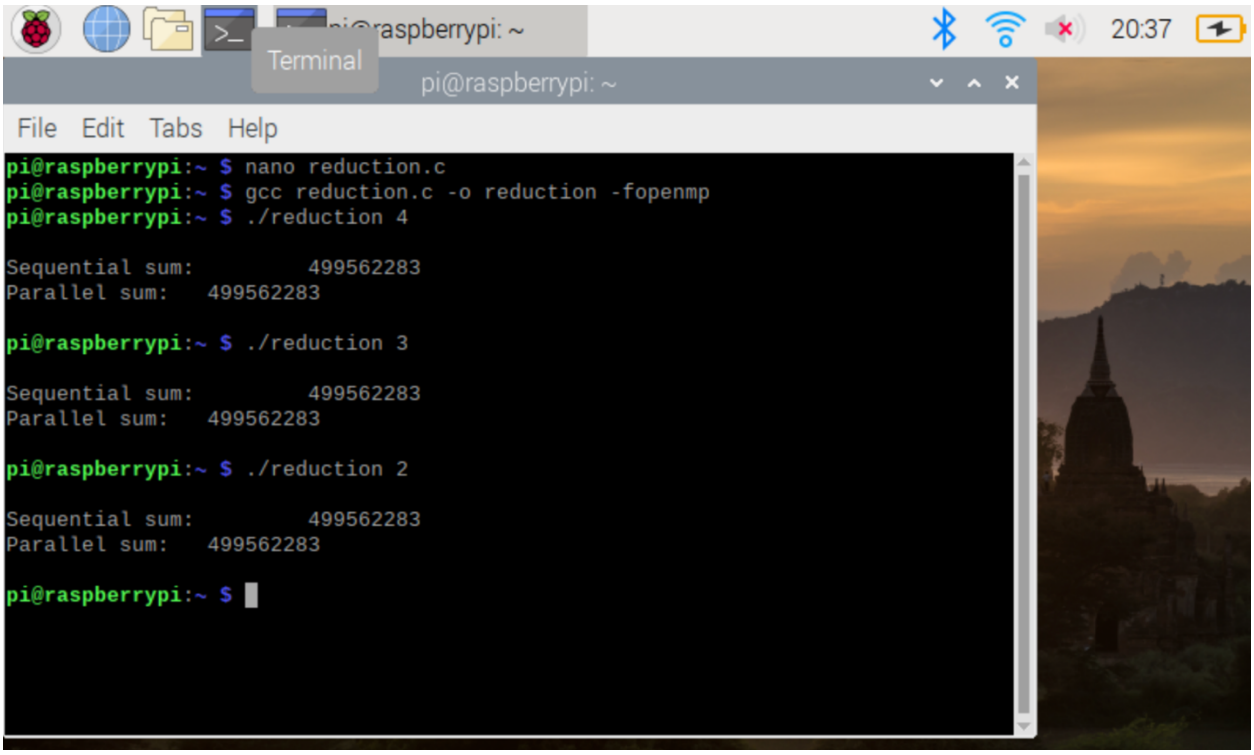


the first thread does not start from thread 0 and goes to thread 3 like static, it changes every time I execute the code since it is dynamic. However, whichever thread started will be the first to receive the value after all four threads have their own 3 iterations like usual.

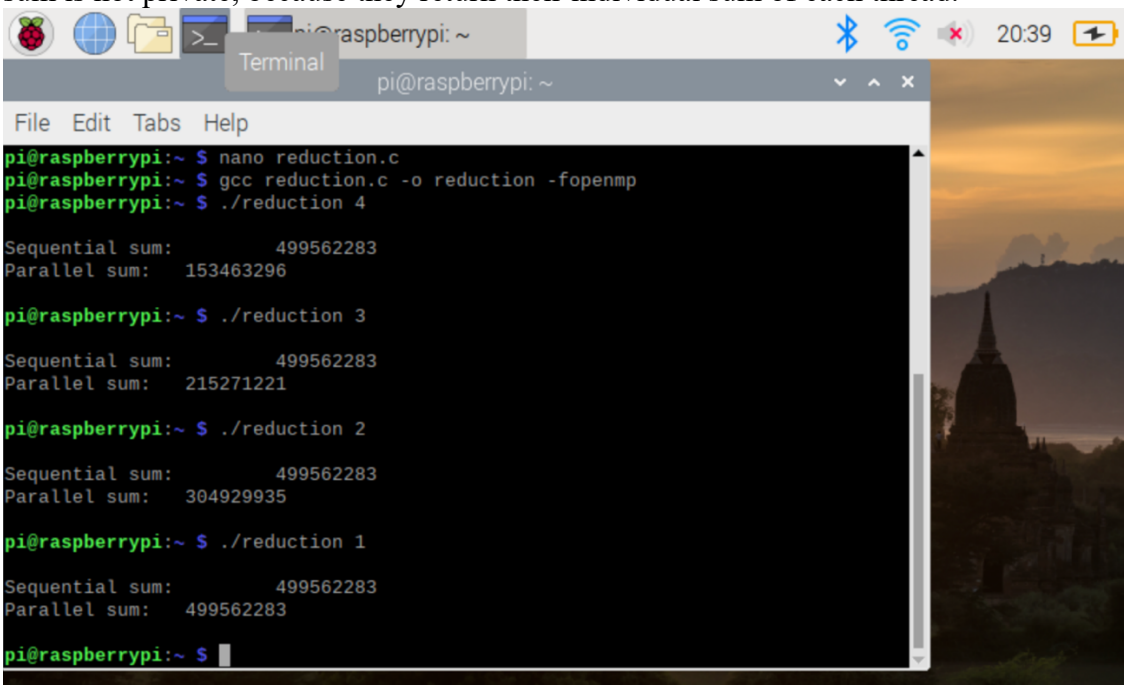




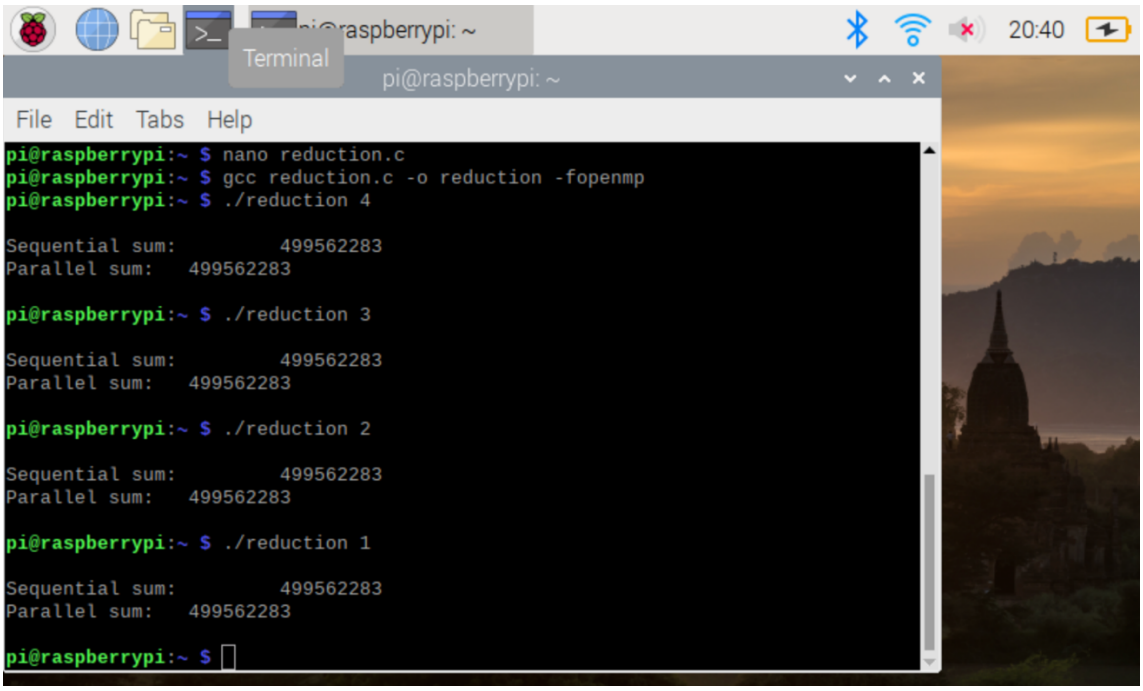
Next, I examined what happens when **loops have dependencies**. After running the program, the output of sequential sum and parallel sum are the same with 4 or different number of threads, because the statement #pragma omp parallel for reduction(+:sum) is commented in the parallelSum function.



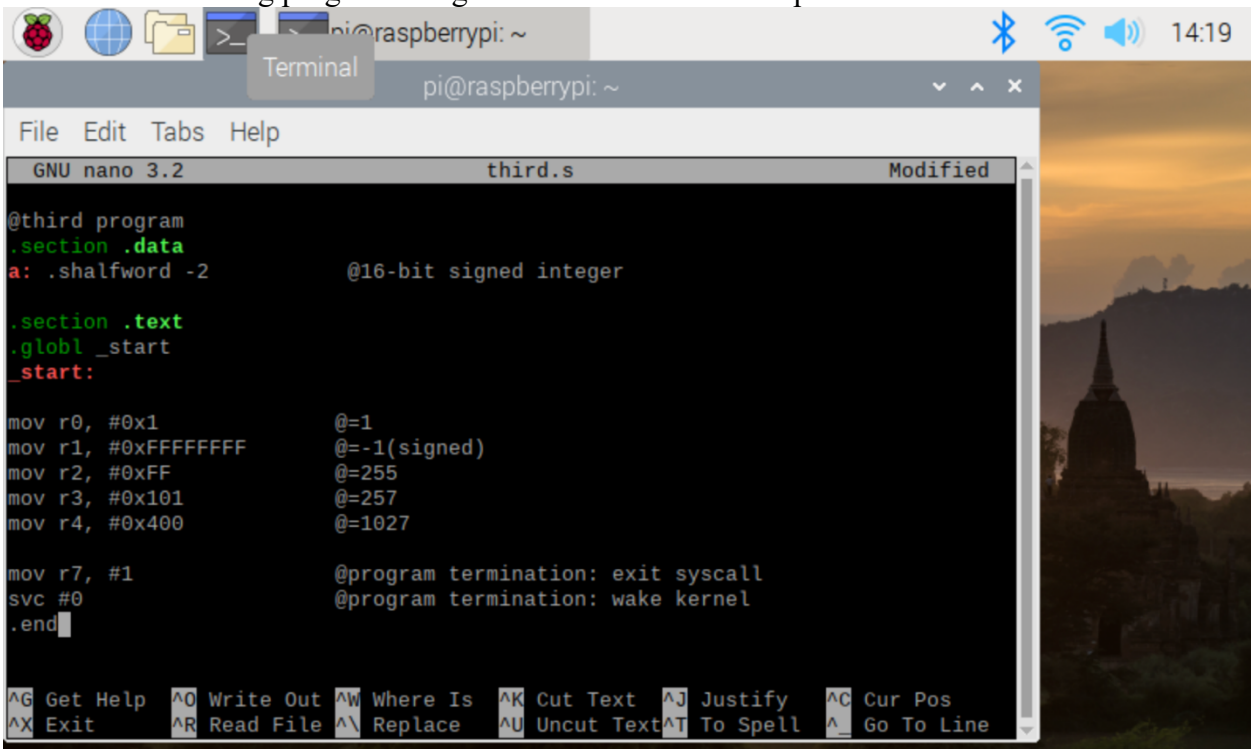
I tried removing the first comment. Now the results are different, Because the accumulator called sum is not private, because they return their individual sum of each thread.



Then I tried removing the second comment. The results were same now. By removing the comment from the reduction clause, the accumulator is made private to each thread and sum up their individual sums.

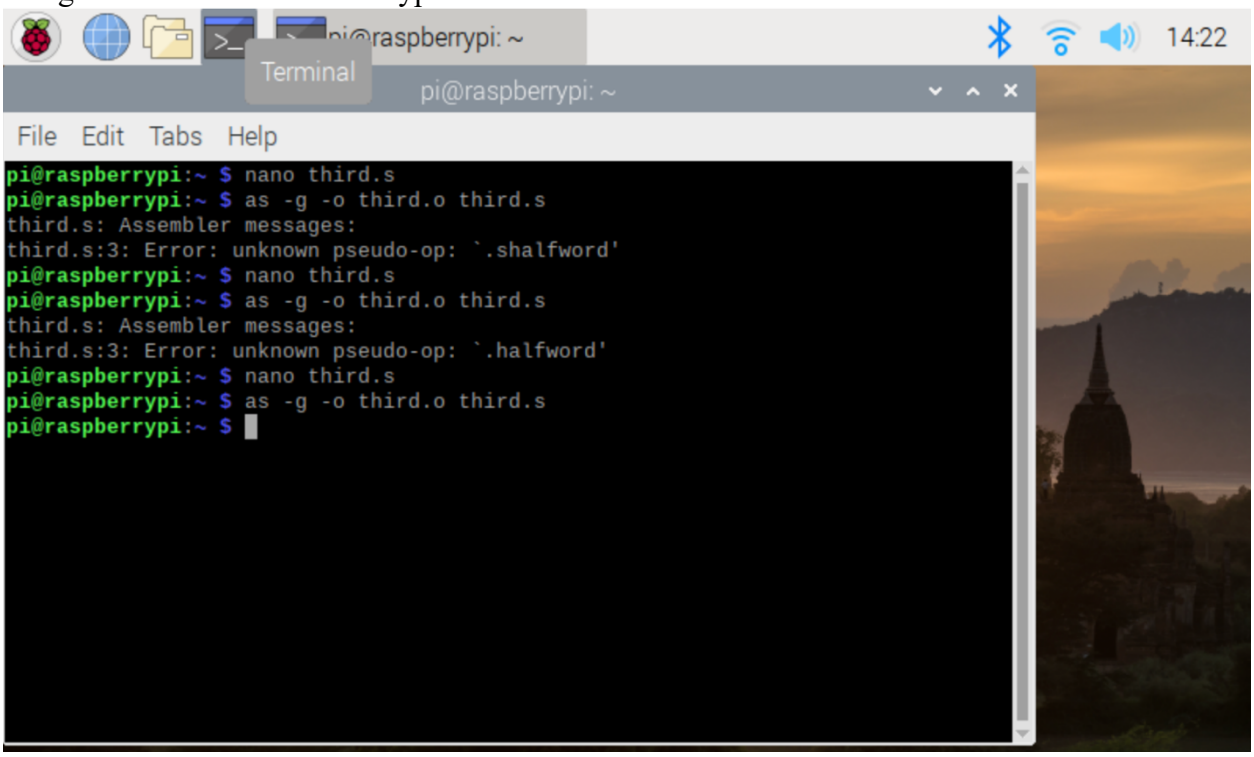


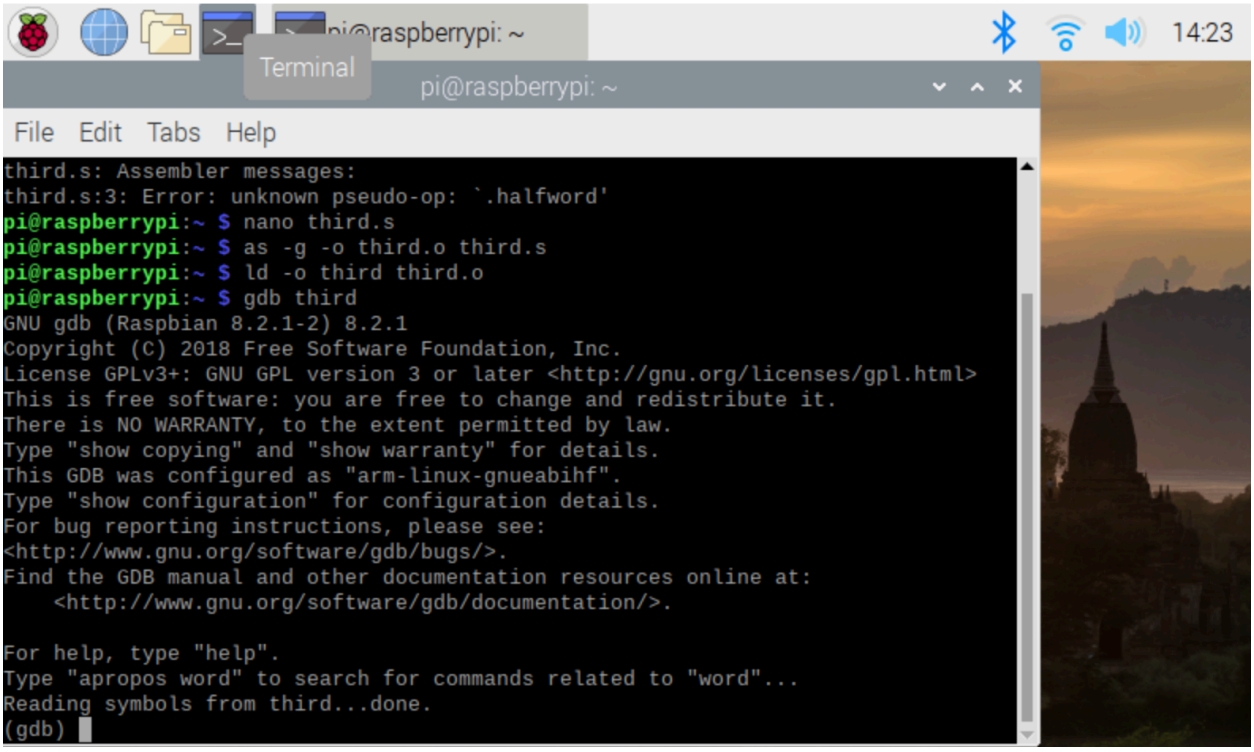
TASK 4: Arm assembly programming.
Part 1: Third Program
I wrote the following program using nano text editor and compiled it



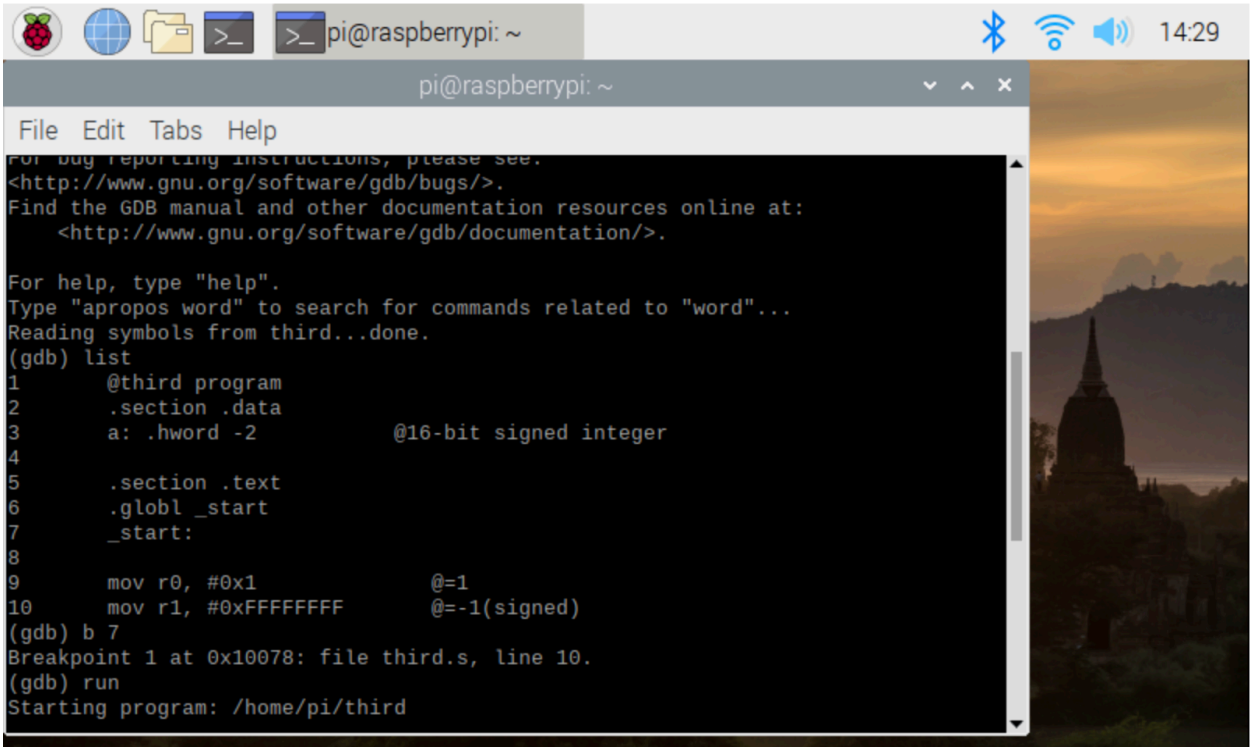
There were two errors: an “unknown pseudo-op’” and a “bad instruction” error. And then I modified the code, changing .shalfword to .hword because the ARM assembly does not

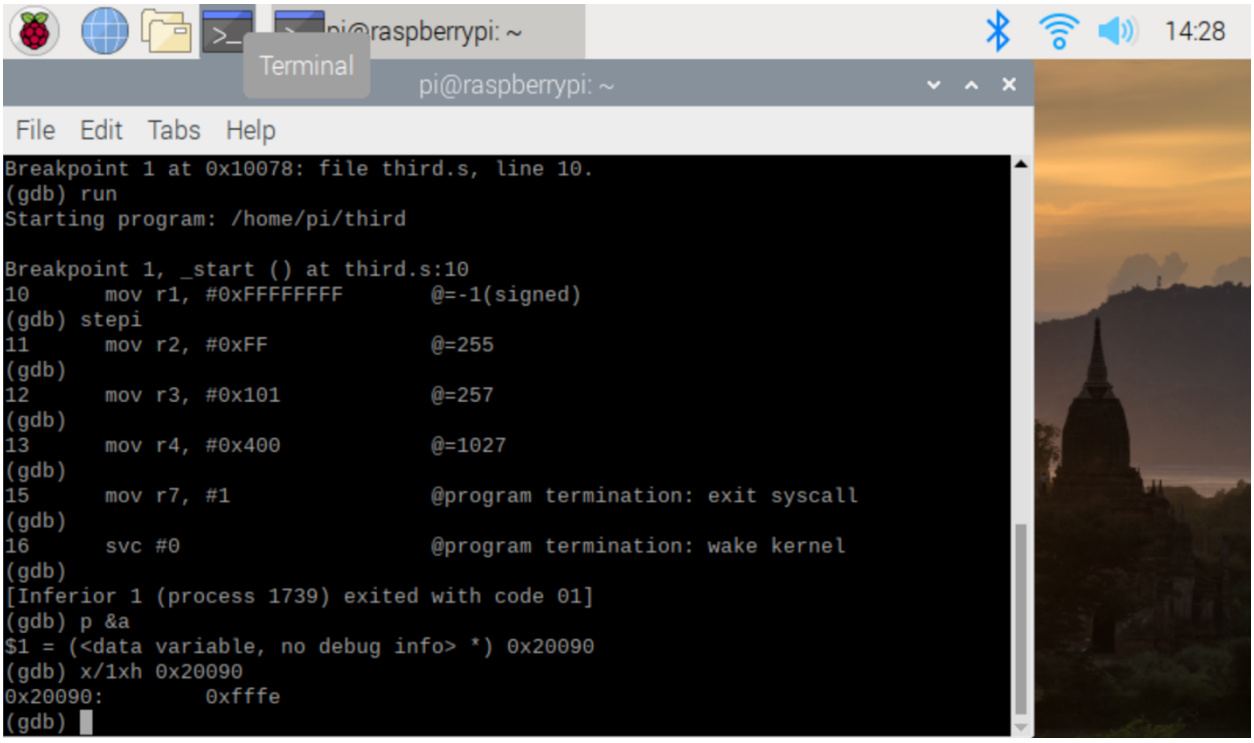
recognize shalfword as a data type.



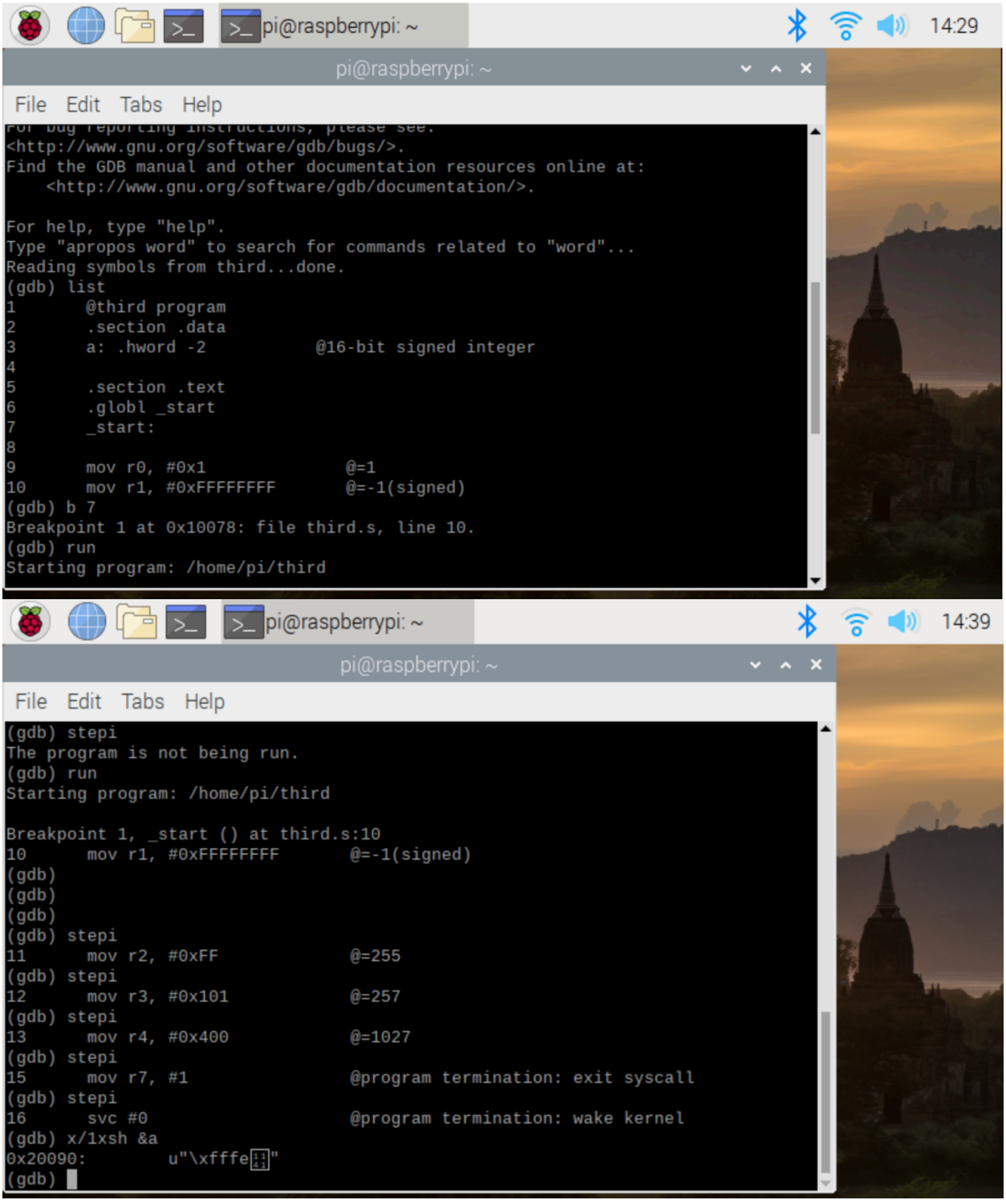


Then I assembled linked and debugged the code using gdb. Then I set a breakpoint at line 7 to debug the code.



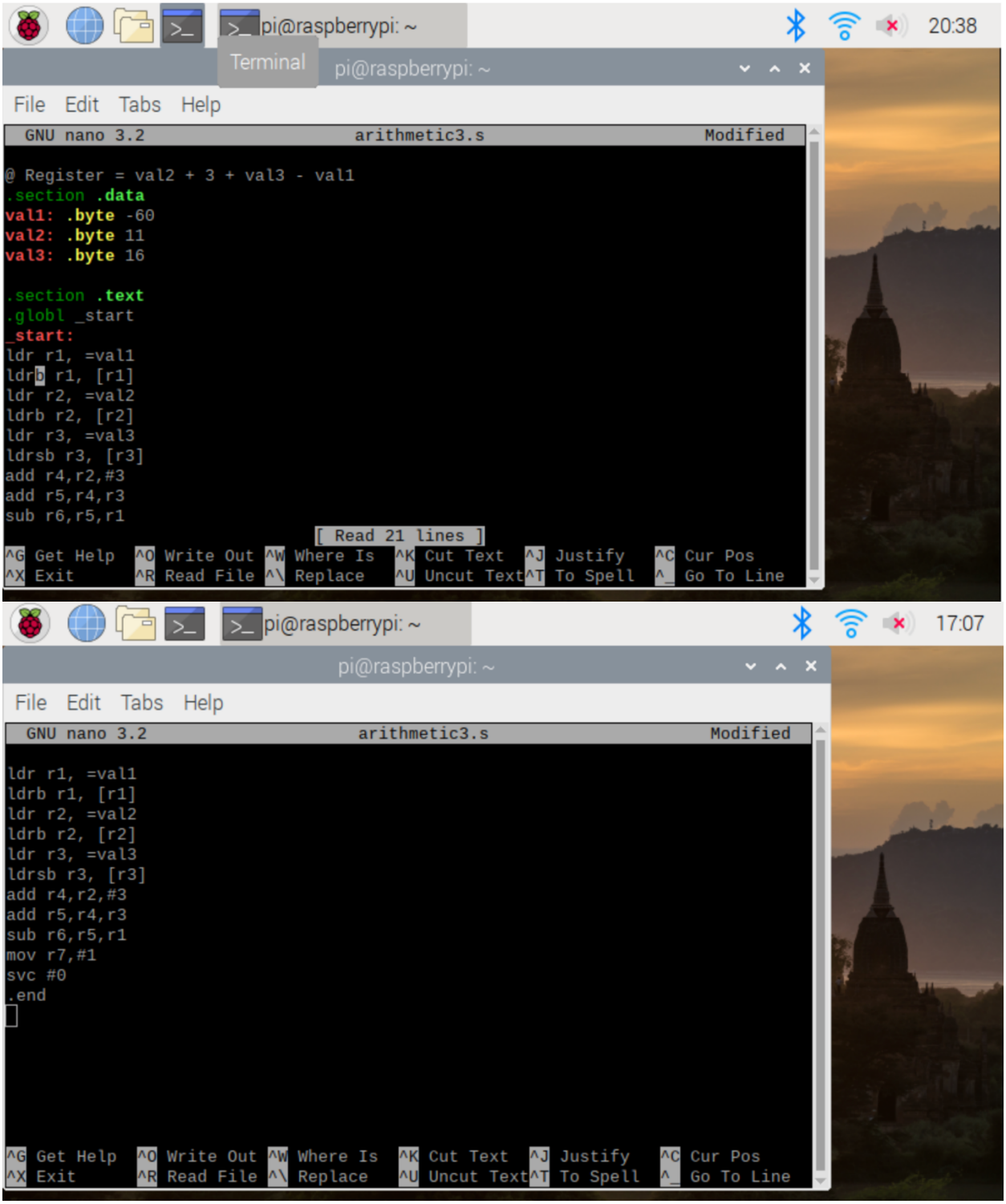


If we use x/1xh it returns FFFE, which is -2 in hexadecimal.

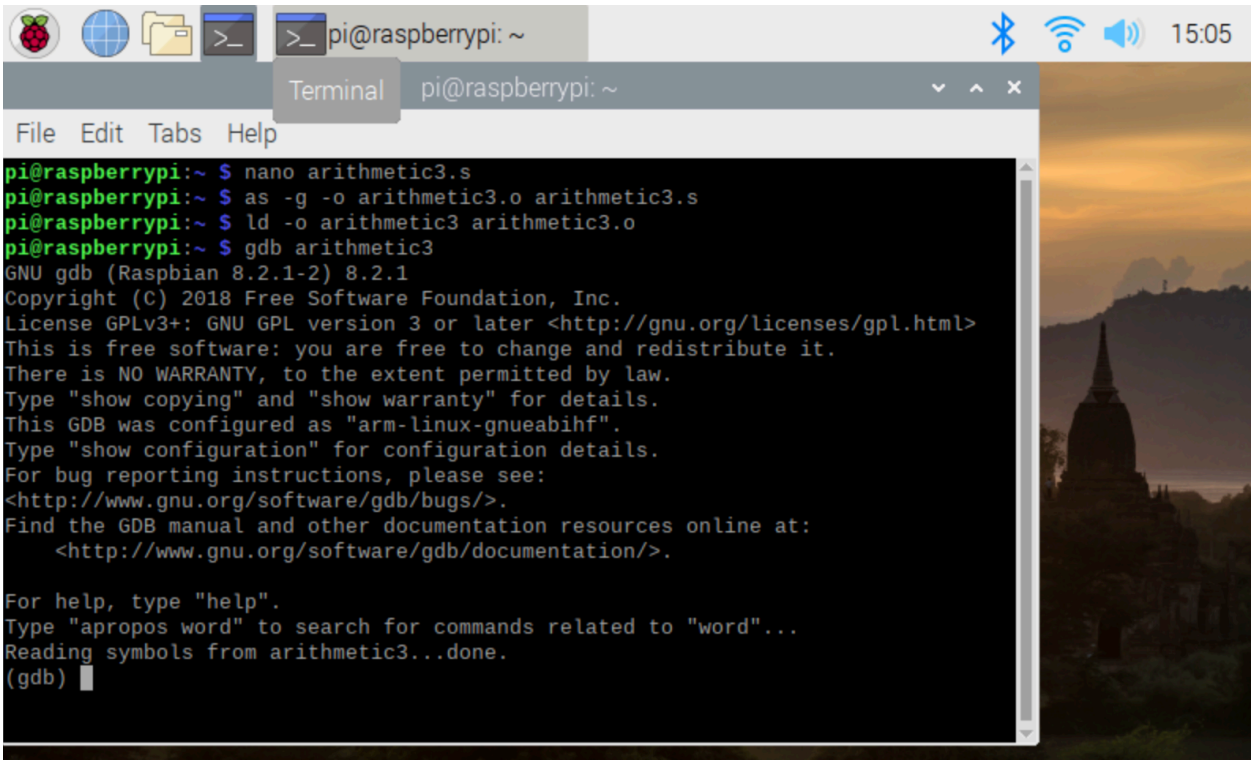


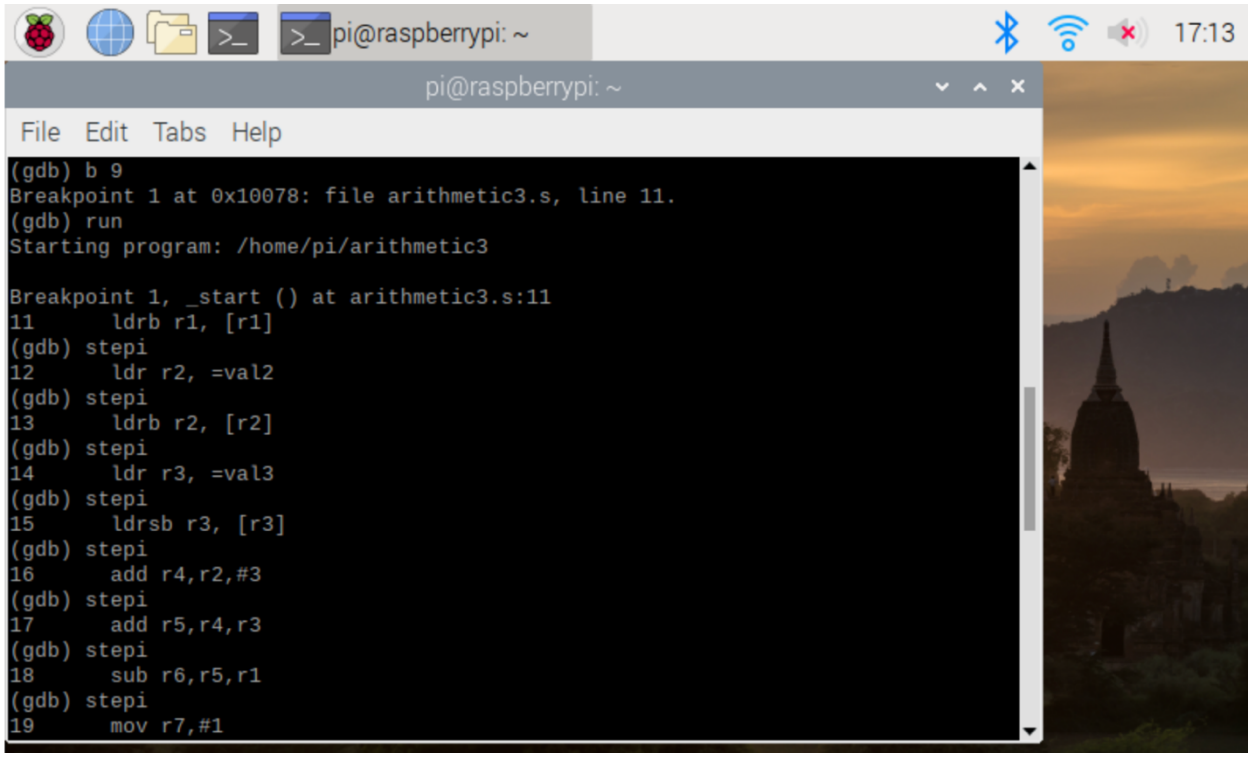
Part 2: Calculate the Expressions

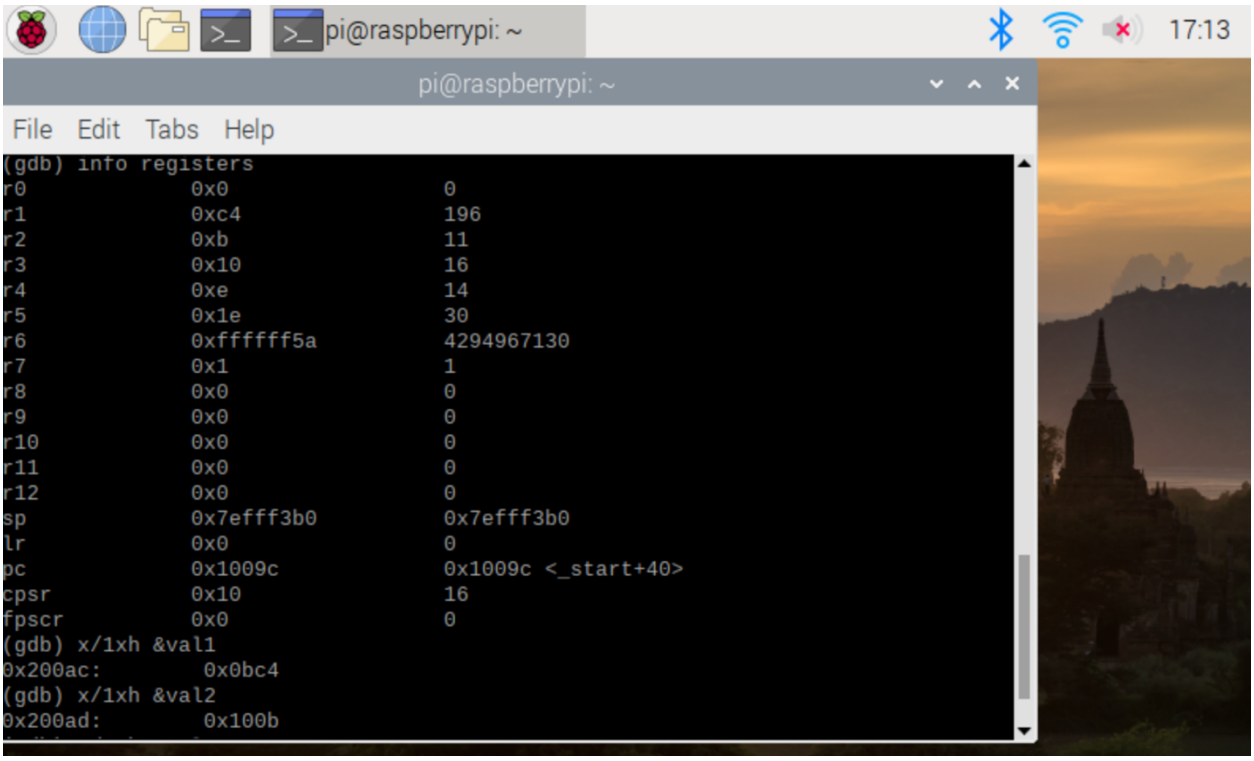
I created the file named arithmetic3 for the following code then I assembled, executed, linked, ran, and debugged the program which calculated the expression: Register = val2 + 3 + val3 – val1, where val1=-60, val2=11, and val3=16. with nano text editor, before assembling it.



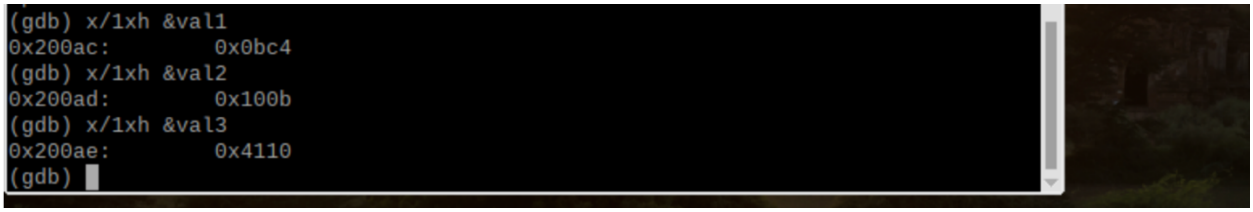
I used gdb to debug the code and set a breakpoint at line 9. Then ran the code and used stepi (step by step) to look through each and every line of the code.


pi@raspberrypi: ~
15:05
Terminal
pi@raspberrypi: ~
File Edit Tabs Help
pi@raspberrypi:~ $ nano arithmetic3.s
pi@raspberrypi:~ $ as -g -o arithmetic3.o arithmetic3.s
pi@raspberrypi:~ $ ld -o arithmetic3 arithmetic3.o
pi@raspberrypi:~ $ gdb arithmetic3
GNU gdb (Raspbian 8.2.1-2) 8.2.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "arm-linux-gnueabihf".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from arithmetic3...done.
(gdb)


pi@raspberrypi: ~
17:13
pi@raspberrypi: ~
File Edit Tabs Help
(gdb) b 9
Breakpoint 1 at 0x10078: file arithmetic3.s, line 11.
(gdb) run
Starting program: /home/pi/arithmetic3
Breakpoint 1, _start () at arithmetic3.s:11
11 ldrb r1, [r1]
(gdb) stepi
12 ldr r2, =val2
(gdb) stepi
13 ldrb r2, [r2]
(gdb) stepi
14 ldr r3, =val3
(gdb) stepi
15 ldrsb r3, [r3]
(gdb) stepi
16 add r4,r2,#3
(gdb) stepi
17 add r5,r4,r3
(gdb) stepi
18 sub r6,r5,r1
(gdb) stepi
19 mov r7,#1



The values in the registers are the same with the value we assign and the value we need to compute: val2 + 3 + val3 -val1 = 11 + 3 + 16 – (-60) = 90.
r1 = val1 = C4 = 60; r2 = b = val2 = 11; r3 = val3 =16; r4 = r2 +3 = 14; r5 =r4 + r3 = 30; r6 = 5a = 90



Then I checked the the values in the memory. The values in the memory are as expected, val1 = c4, val2 =0b, val3 =10 which in decimal is -60, 11 and 16.

We I checked the negative flag value, Since CPSR return 00000010 in hexadecimal, the negative flags is the 31 bits of the value and it equals to 0, which is true because the result is not a negative number.

Zoe Kosmicki

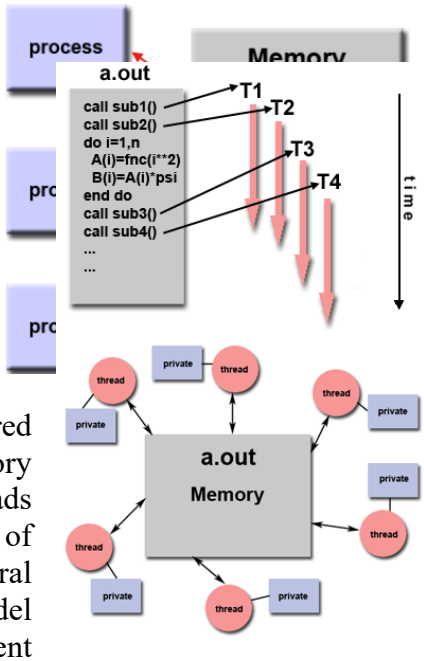
TASK 3: Parallel programming skills

Part 1: Foundation

- *Define the following:*
  - Task: a set of instructions that are generally organized like a program, to be executed by a processor.
  - Pipelining: the way a program is broken down by various processors to be executed in a parallel fashion. Comparable to an assembly line.
  - Shared Memory: a type of computer architecture that lets all processors have direct access to memory. This isn't dependent on where the physical memory is located; each processor has the same access to the logical memory.
  - Communications: how parallel tasks "talk" to each other as they are being executed.
  - Synchronization: how a parallel program coordinates with itself as it's running.
- *Classify parallel computers based on Flynn's taxonomy. Briefly describe every one of them.*
  - SISD (Single Instruction stream; Single Data stream): SISD computers are serial, so they can only process and execute one data/instruction stream at a time. Generally now only found in older hardware.
  - SIMD (Single Instruction stream; Multiple Data stream): Parallel computer that can execute the same instruction over various data threads at the same time. The two main varieties are processor arrays and vector pipelines.
  - MISD (Multiple Instruction stream; Single Data stream): A type of parallel computer that executes more than one set of instructions on one stream of data at the same time. This is the most common type of parallel computer currently.
  - MIMD (Multiple Instruction stream; Multiple Data stream): A parallel computer that executes multiple instruction and data streams at the same time. This type of computer makes up most modern supercomputers and network clusters.
- *What are the Parallel Programming Models ?*
  - Shared memory, threads, distributed memory, data parallel, hybrid, SPMD, and MPMD
- *List and briefly describe the types of Parallel Computer Memory Architectures. What type is used by OpenMP and why?*
  - UMA (Uniform Memory Access) and NUMA (Non-Uniform Memory Access) are the types of Parallel Computer Memory Architectures. UMA machines have one shared general memory that multiple CPUs have access to, while NUMA machines

have several memories connected to several CPUs, which are all interconnected via buses. OpenMP can use both, since it is built for shared memory machines.

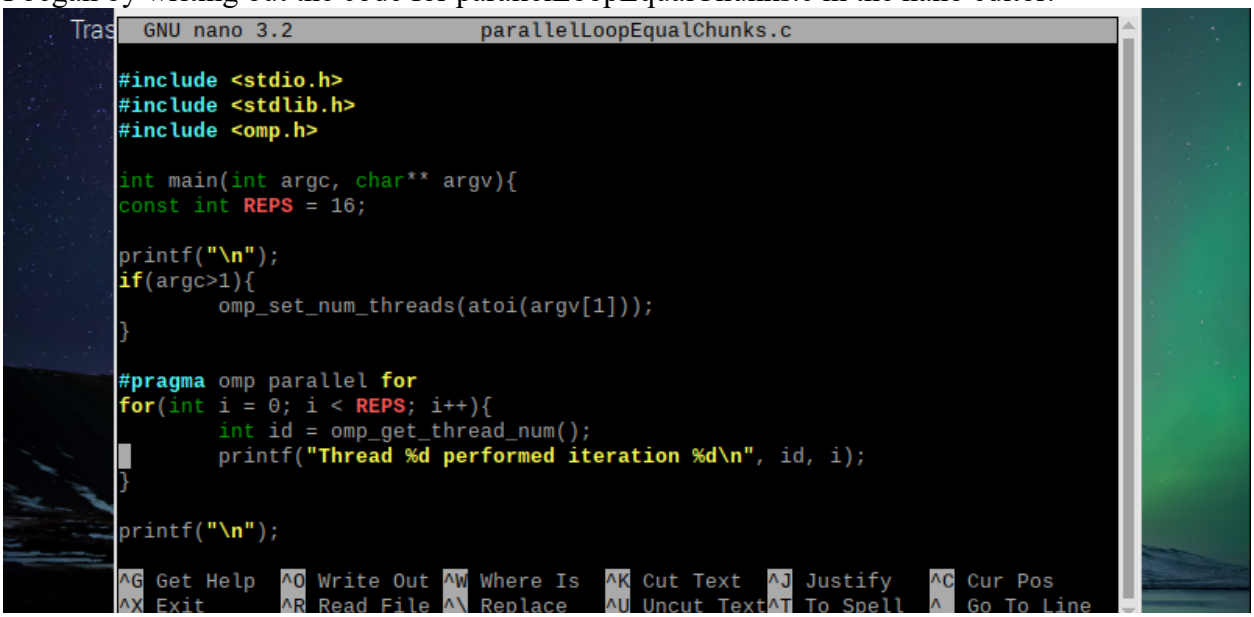
- *Compare Shared Memory Model with Threads Model? (in your own words and show pictures)*
  - Shared Memory Model: the memory is a shared space, with each individual process able to read and write to shared addresses within the memory.
  - Threads Model: a program has its main body that allocates resources to run the program in full. After the resources have been found and set aside, the main body creates several threads/tasks that operate independently and in parallel. These threads all operate within a shared memory, so they can all see what the other threads are doing and have access to the resources other threads might be using.
    - These models are both forms of parallel processing, and they both utilize a shared memory space. However, the shared memory model is much simpler than the threads model. The shared memory model consists of just a shared memory space that several processes share, while the threads model utilizes a shared memory allocated by a parent process, and only accessed by threads from within the parent process.



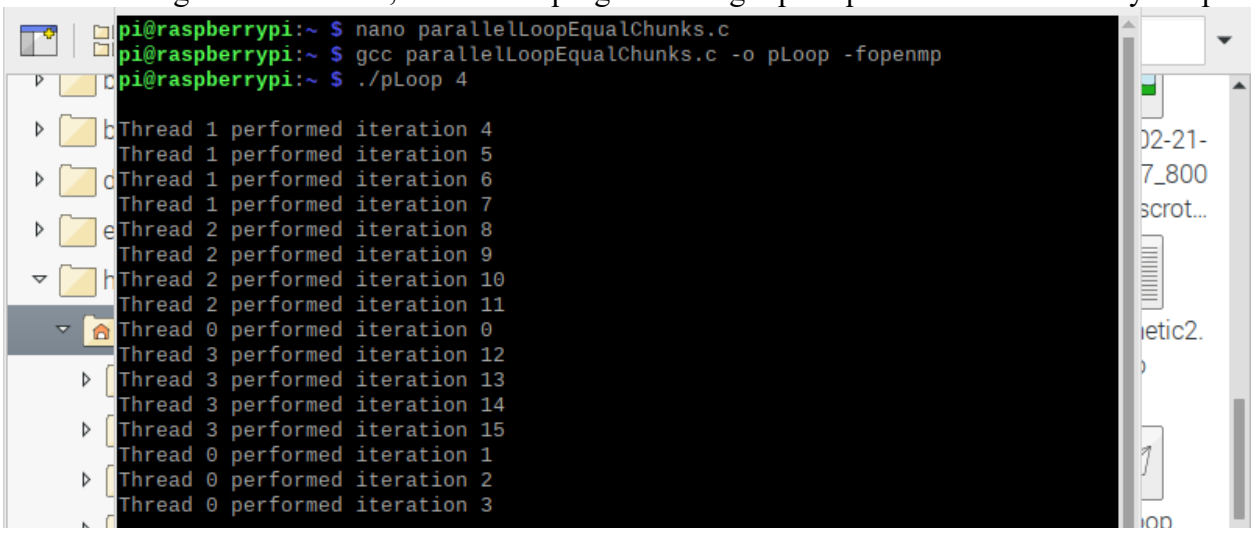
- *What is Parallel Programming? (in your own words)*
  - Programming that takes advantage of a computer's ability to complete tasks in parallel with each other rather than just serially.
- *What is system on chip (SoC)?*
  - A type of computer that has the CPU, GPU, and RAM put onto a single chip, instead of all of those components separately placed
- *Does Raspberry PI use SoC?*
  - Yes.
- *Explain what the advantages are of having a System on a Chip rather than separate CPU, GPU and RAM components.*
  - SoCs are smaller than having separate components, so they are better for smaller devices as well as are cheaper to make due to its size. They also generally use less power, since wired information processing is literally/physically shorter compared to a PC motherboard.

Part 2: Parallel Programming Basics

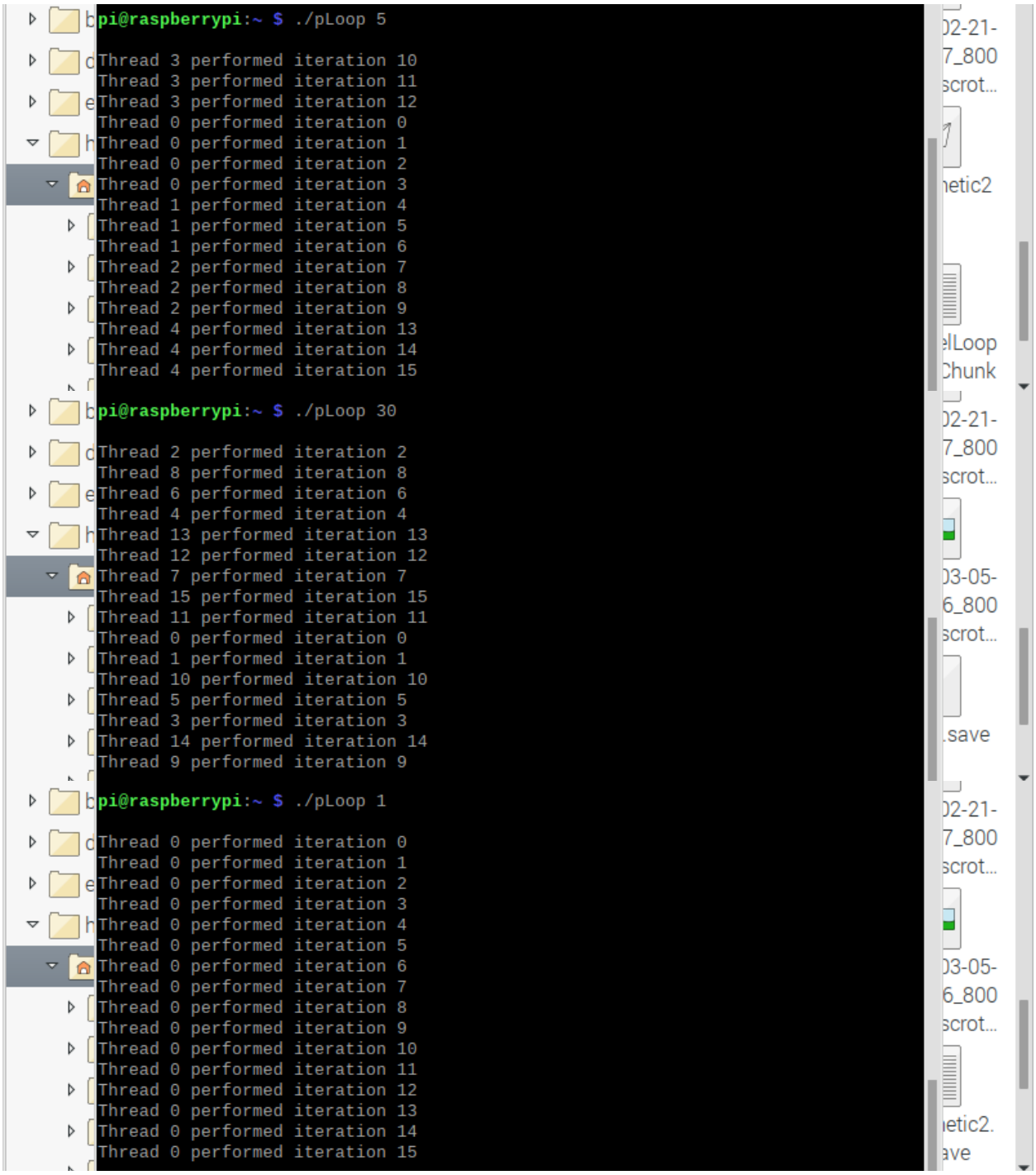
I began by writing out the code for parallelLoopEqualChunks.c in the nano editor.



After creating an executable, I ran the program using ./pLoop 4 and below is my output.

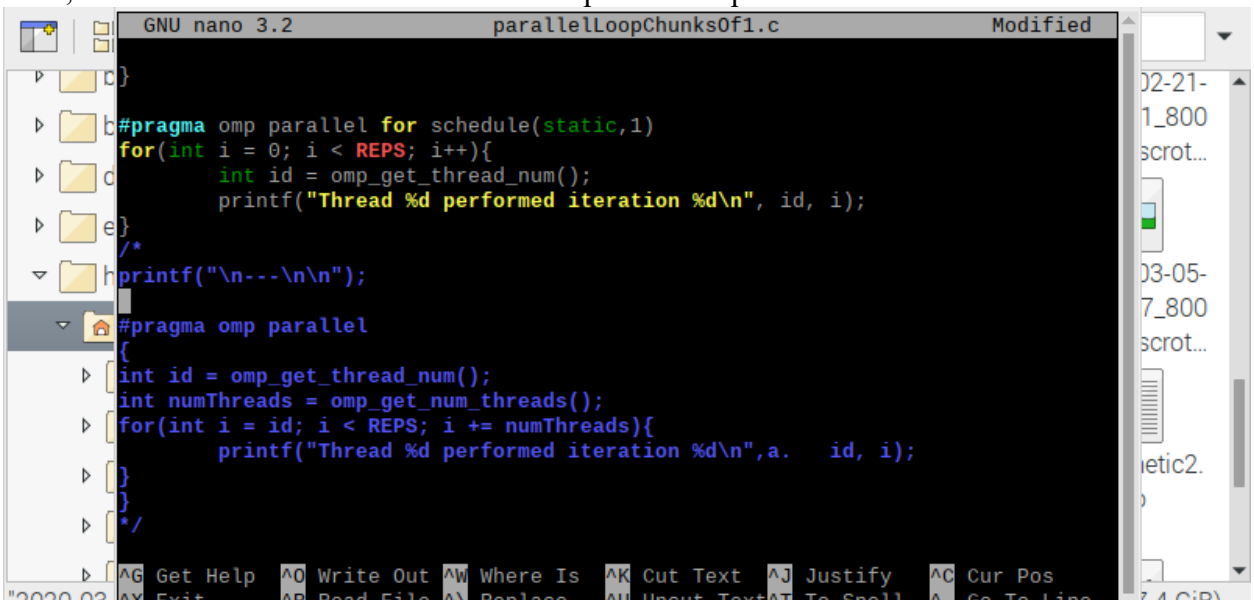


I then tried a few other numbers of threads to fork to see what the output would look like.

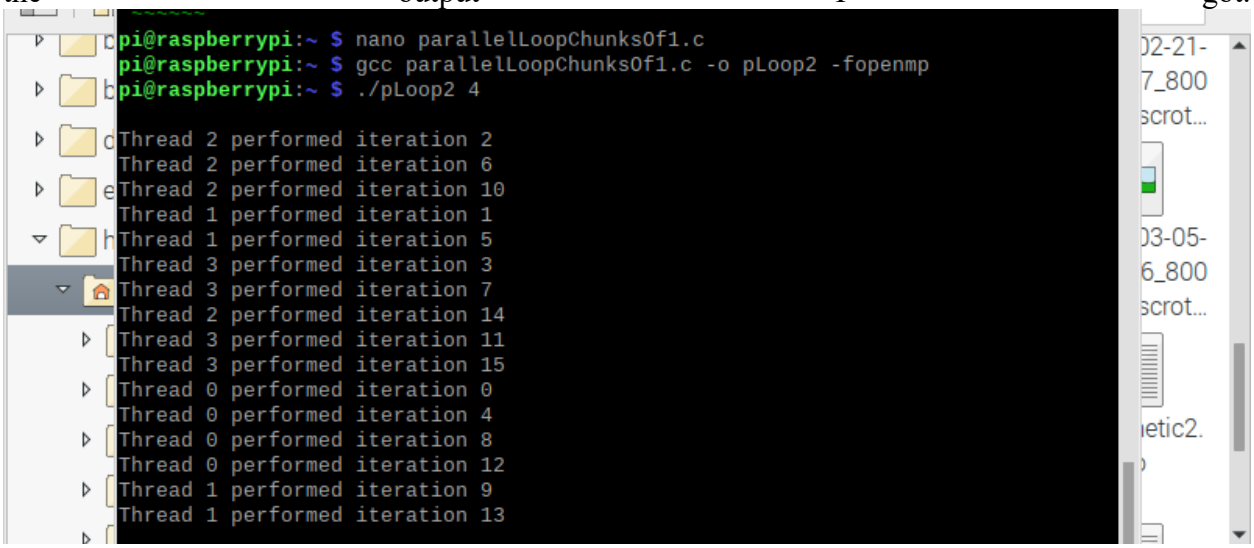


To me, it seemed like if there are enough threads (at least 16), then the iteration would pair to it's similarly numbered thread. Otherwise, there would be a sectioning of threads that would attempt to partition each iteration as cleanly as possible.

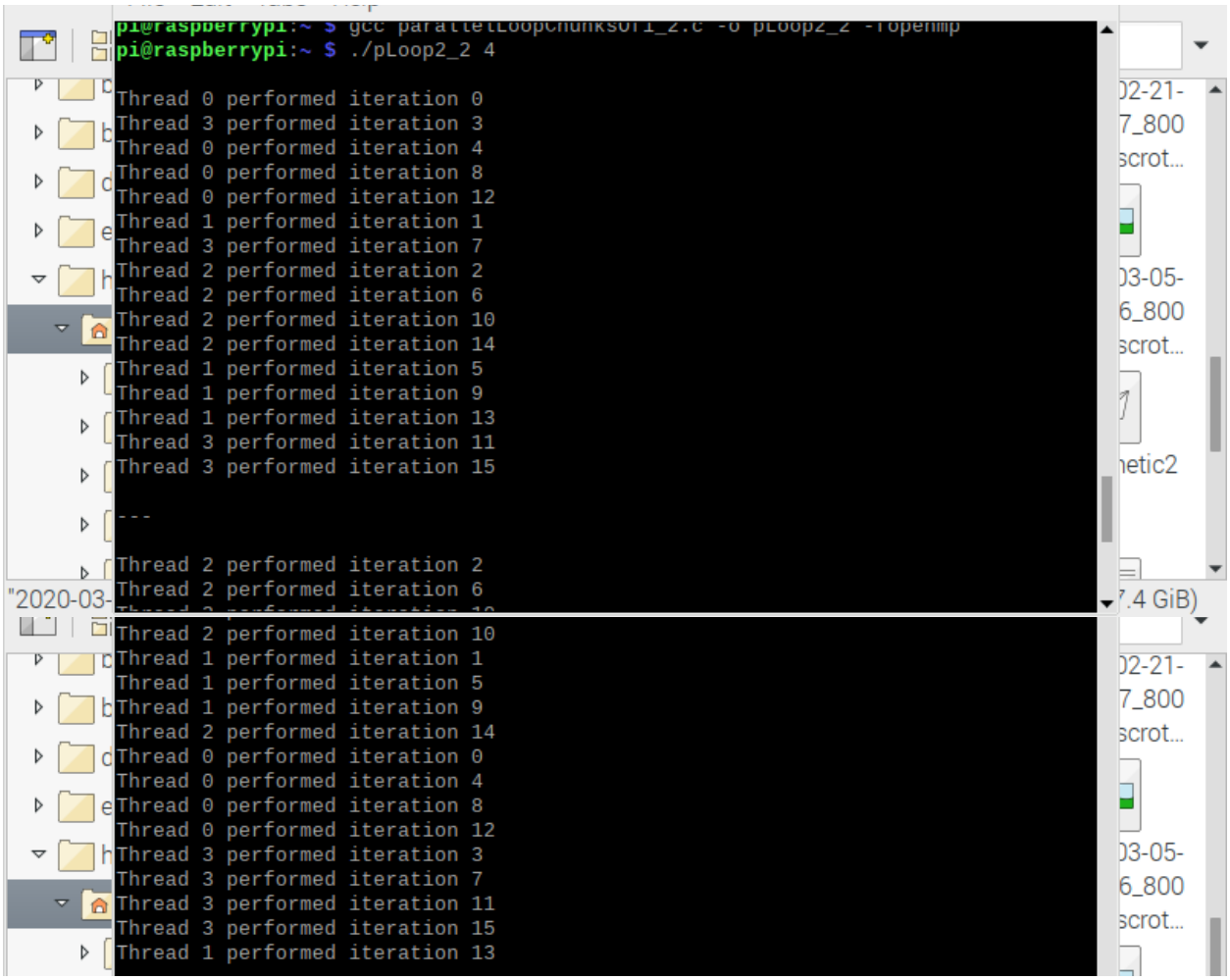
Next, I wrote down the code for parallelLoopChunksOf1.c in the nano editor.



I first ran pLoop2 with the pragma section commented out, just to make sure it ran fine. Below is the output I got.



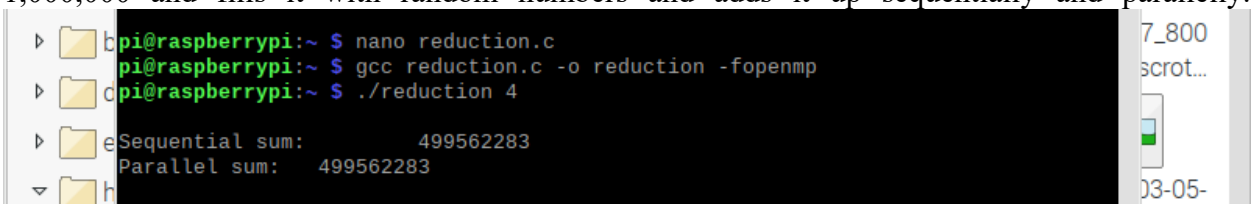
After running that, I uncommented the pragma section and renamed the executable pLoop2_2, just to keep track of it. Then I ran it with the command ./pLoop2_2 4 and below is the output.



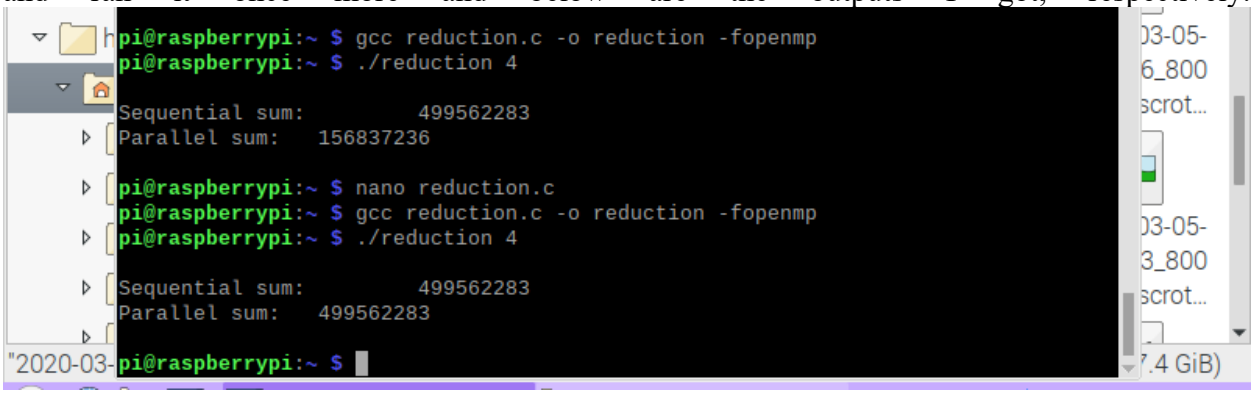
Both sections of code give equal outputs, just in different orders. The second chunk of code seems more flexible than the first, because in the for loop, i is set to id instead of starting at 0, meaning it's easier to adapt the code to a different number of threads.

Next, I wrote out the reduction.c program in the nano editor.

After compiling and linking it, I ran it with the value 4. Both the sequential and parallel sums were the same, since the parallel portion was still commented out. This program creates an array of size 1,000,000 and fills it with random numbers and adds it up sequentially and parallelly.



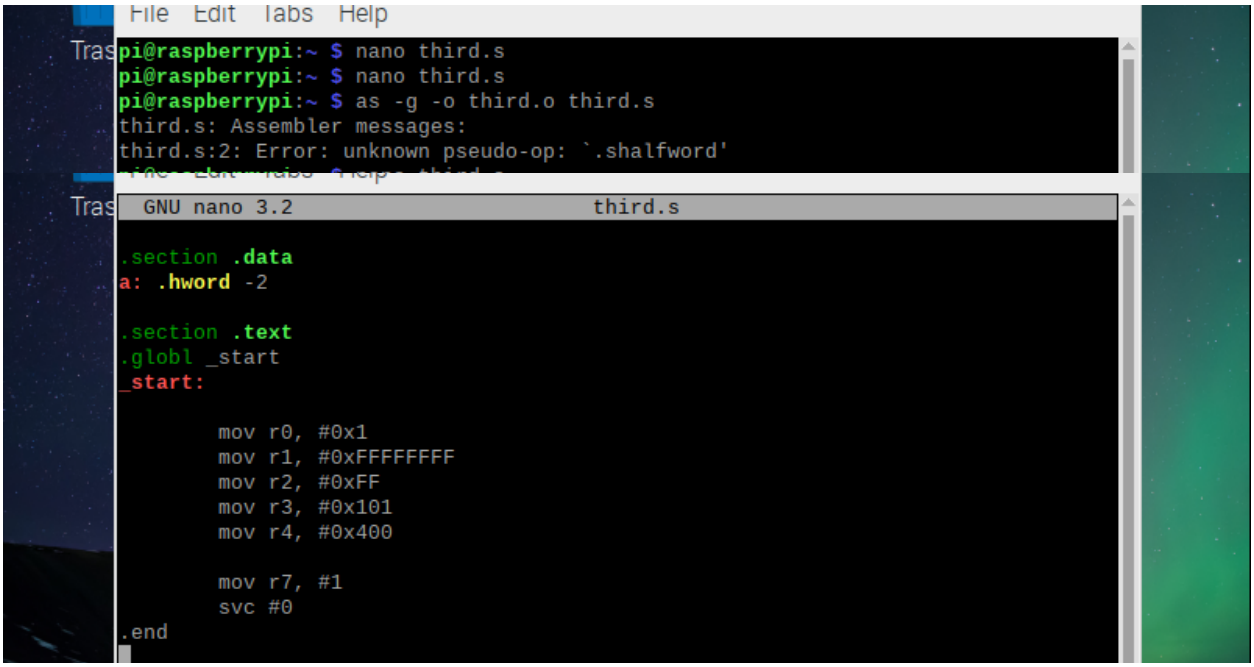
Next, I uncommented line 39, ran the program again, and then uncommented reduction(+:sum) and ran it once more and below are the outputs I got, respectively.



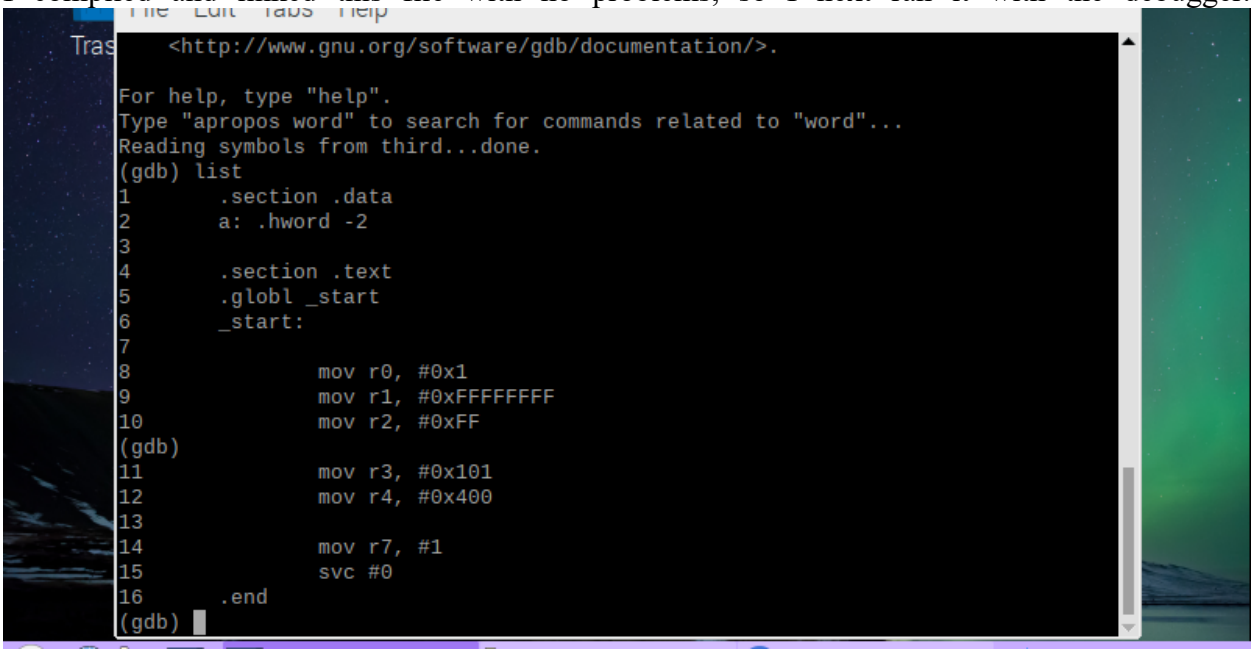
Before the reduction(+:sum) clause was uncommented, the parallel and sequential sums were different. After adding the reduction clause back in, they went back to being the same value. I think they were different at first because without the reduction clause, the parallel processes weren't communicating with each other correctly, with many different sum variables being calculated and not added together at the end. This would end up giving a different answer than the sequential sum would give.

TASK 4: Arm assembly programming.

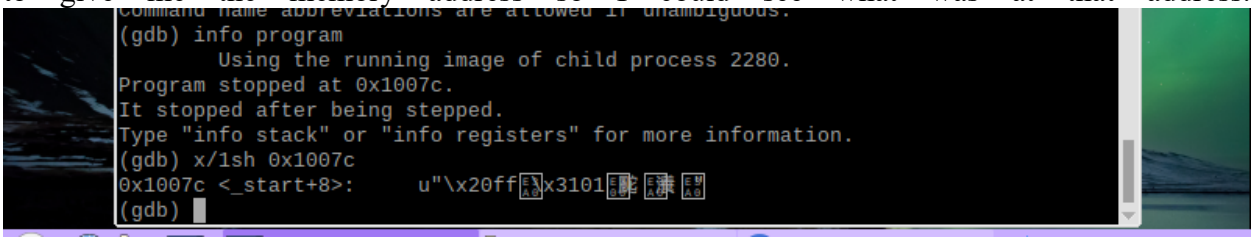
I started this task by writing out third.s in the nano editor, and then trying to compile and link it. I ran into the error message "third.s:3: Error: unknown psuedo-op: `.shalfword'". This makes sense, because the compiler doesn't understand "shalfword", so I changed it to a: .hword -2.



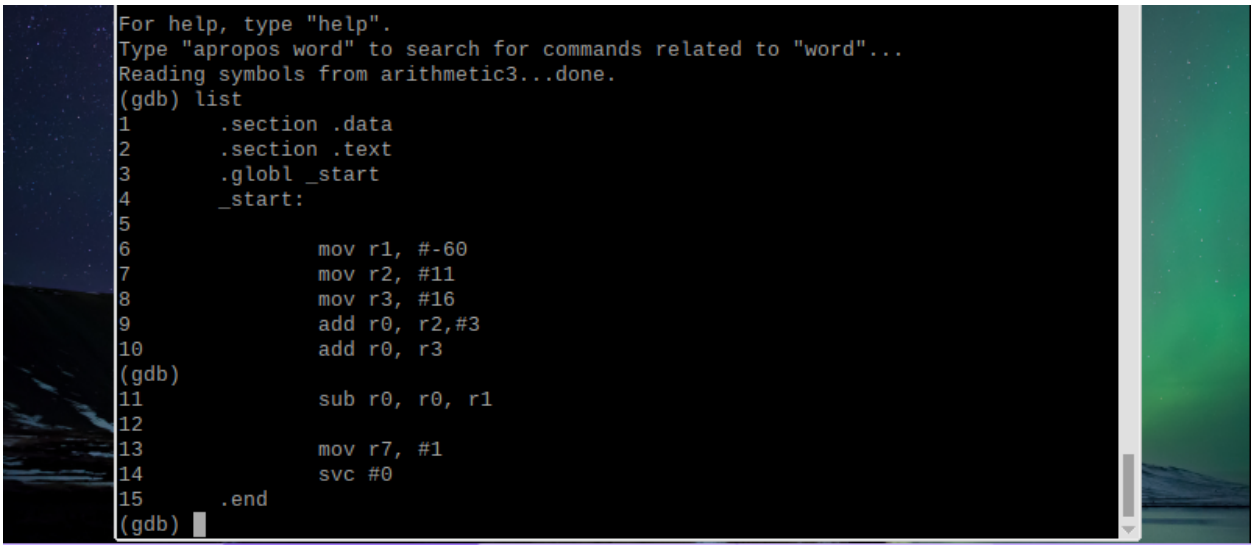
I compiled and linked this file with no problems, so I next ran it with the debugger.



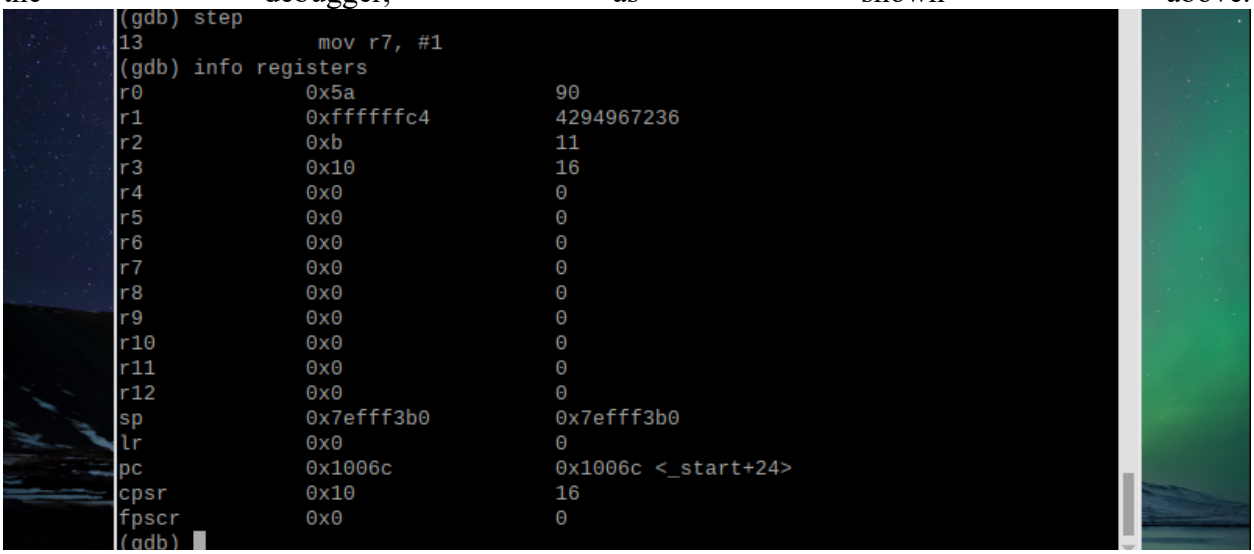
I added a breakpoint in at line 8 and ran it. After taking a step, I used the info program instruction to give me the memory address so I could see what was at that address.



As shown above, it gave me the address and offset from the start, along with some indecipherable characters.



Next, I wrote out the arithmetic3.s program in the nano editor, compiled it, linked it, and ran it in the debugger, as shown above.



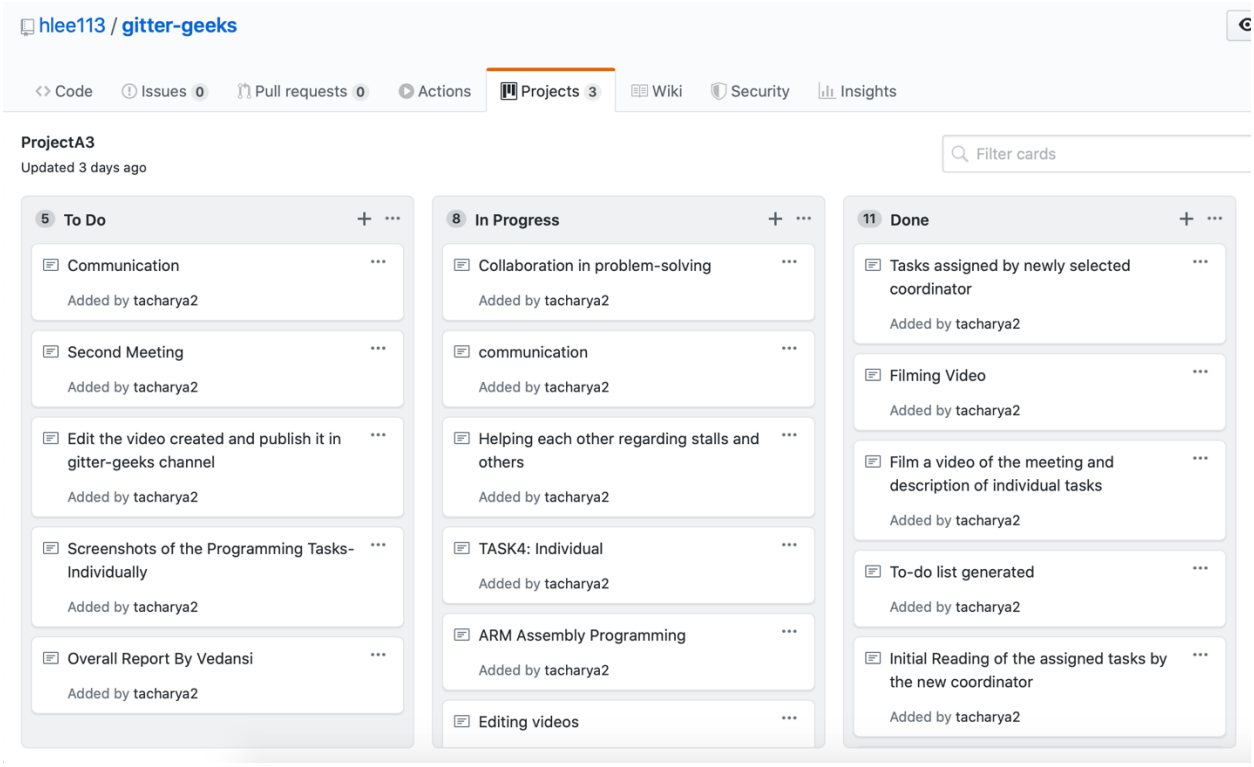
I added a breakpoint at line 6 and stepped through the program, checking the registers right before the exit call. In r0, the value 5A is stored, which is the arithmetic evaluation of 11+3+16-(-60). In r1, there's the 2's complement of -60, which in hex is FFC4. So in decimal it's a huge number, but in arithmetic calculations, it's treated as -60. We can also see that the CPSR register is set to 0x10. This means that the negative flag has been set because we did signed arithmetic.

Appendix: (LINKS)

Slack: https://app.slack.com/client/TN46XP41L/GSUL94RJ5

Github: https://github.com/hlee113/gitter-geeks/tree/master/A3

Youtube: https://www.youtube.com/watch?v=9OWYozh1OWs&amp=&feature=youtu.be



Branch: master ▾ **gitter-geeks / A3 /** Create new file | Upload files | Find file | History

mknight23 Michael Knight — Latest commit 3fa4c47 38 minutes ago

..

A3 Task 3.docx	Michael Knight Task 3	1 hour ago
A3Hyoungjun Assemblyprogramming.docx	Add files via upload	10 hours ago
A4 Tesk 4.docx	Michael Knight	38 minutes ago
Hyoungjun A3Parallel programming skills.docx	Add files via upload	11 hours ago
Kosmicki Task 3 A3.docx	Add files via upload	15 hours ago
Kosmicki Task 4 A3.docx	Add files via upload	15 hours ago
Project A3-Task4-Tek.docx	Add files via upload	12 hours ago
Project A3. 3a-Tek.docx	Add files via upload	4 days ago
Project A3. 3b-Tek.docx	Add files via upload	5 days ago
ex	Create ex	14 days ago