Developing Soft and Parallel Programming Skills Using Project-Based Learning
Spring 2020

Gitter-geeks

Project A4

Team members

Hyoungjun Lee- Team coordinator
Michael Knight
Tek Acharya
Vedansi Parsana
Zoe Kosmicki

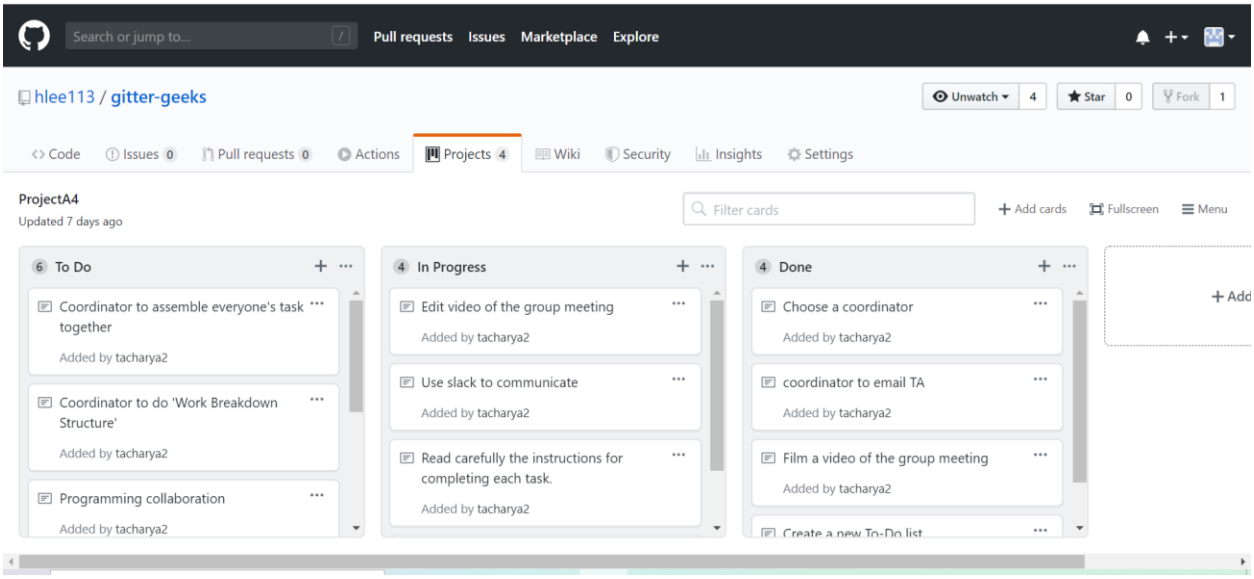
Planning and Scheduling

As Team coordinator, I need to make Planning and Scheduling, since this project held during Spring break, somewhat I had a bit challenge but overall, every team member helps each other and work through together, figured out time for taking video recording, my challenge was nothing. We used Slack to communicate each other with any problem we were having during project. I had never been to assign task to somebody, I just tried to assign task to everyone to be fair. Overall, it was really great time, first time using Zoom to record video, we can keep working more with our group

Task 1

Name	Email	Tasks	Dependency	Duration	Due Date	Note
Hyoungjun Lee (Coordinator	hlee113@student.gsu.edu	-Task 1 -Task 5 -Task 3 -Task 4 -Task 6	Combine everyone's report in 1 report and submitting	4 hours	3/26/20	100%
Michael Knight	mknight23@student.gsu.edu	-Task 3 -Task 4 -Task 6 (Participation)	None	4 hours	3/26/20	100%
Tek Acharya	tacharya2@student.gsu.edu	-Task 2, Github -Task 3 -Task 4 -Task 6 (Participation)	None	4 hours	3/26/20	100%
Vedansi Parsana	vparsana1@student.gsu.edu	-Task 3 -Task 4 -Task 6 (Participation)	None.	4 hours	3/26/20	100%
Zoe Kosmicki	zkosmicki1@student.gsu.edu	-Task 3 -Task 4 -Task 6 , Video editing	None	5 hours	3/26/20	100%

Task 2



Michael Knight

Task 3:

Parallel Programming Skills

A) Foundation

Race Condition

1. What is Race condition?

A race condition or race hazard is where a program's output is dependent on the sequence of or timing of uncontrollable variables.

2. Why race condition is difficult to reproduce and debug?

Since the variables or events are uncontrollable, trying to reproduce the results is unlikely, when debugging the production systems can disappear when additional logging is added or when attaching a debugger.

3. How can it be fixed? Example from project A3:

By Declaring the variable as private it fixes the problem that causes the race condition.

4. Summaries the Parallel Programming Patterns section I the "Introduction to Parallel Computing_3.pdf" (two pages) in your own words.

Parallel Applications branch off in two directions, “Strategies” and “Concurrent execution Mechanisms.” Strategies uses two “Parallel Algorithm Strategy” Data and task decompositions, as well as two “Implementation Strategy” Program and Data Structure, While Concurrent Execution Mechanisms only uses “Process/Thread Control” and “Coordination.”

5. In the section “Categorizing patterns” in the “Introduction to Parallel Computing_3.pdt” compare the following:
    a. Collective synchronization(barrier) with Collective communication (reduction)

A Collective Synchronization (barrier) is like a gate that doesn’t allow progression until everything is finished. Collective Communication (reduction) allows the processes to be reorganized to be performed more efficiently

    b. Master-worker with fork join

Fork join is similar to branching, when traveling on a path and you meet a fork 1 path becomes multiple, Master-worker doesn’t branch off and works in parallel with each worker.

6. Dependency: Using your own words and explanation, answer the following:
    a. Where can we find parallelism in programming?

Basically everywhere, since its way more efficient then programming sequentially, can be found in any type of computers, CPUs, cellphones, and servers.

    b. What is dependency and what are its types (provide one example for each)?

A Dependency is when an operation depends on another in order to complete and produce a result.

True dependences (s1: a=1; S2: b=a;), Anti-dependences (S1: a=b; S2: b=1;), Output dependences (S1: a=f(x); S2: a=b;)

    c. When a statement is dependent and when it is independent (provide two examples)?

A statement is dependent when it has to be preformed sequentially, meaning in order to go to statement 2, statement 1 has to be executed. (example) S1: a=1; S2: b=a;

Independent is where both statements can run in any order without having to wait on the other to finish. (example) S1: a=1; S2: b=2;

    d. When can two statements be executed in parallel?

When each statement is Independent.

    e. How can dependency be removed?

By Rearranging statements or eliminating statements

    f. How do we compute dependency for the following two loops and what types of dependency?

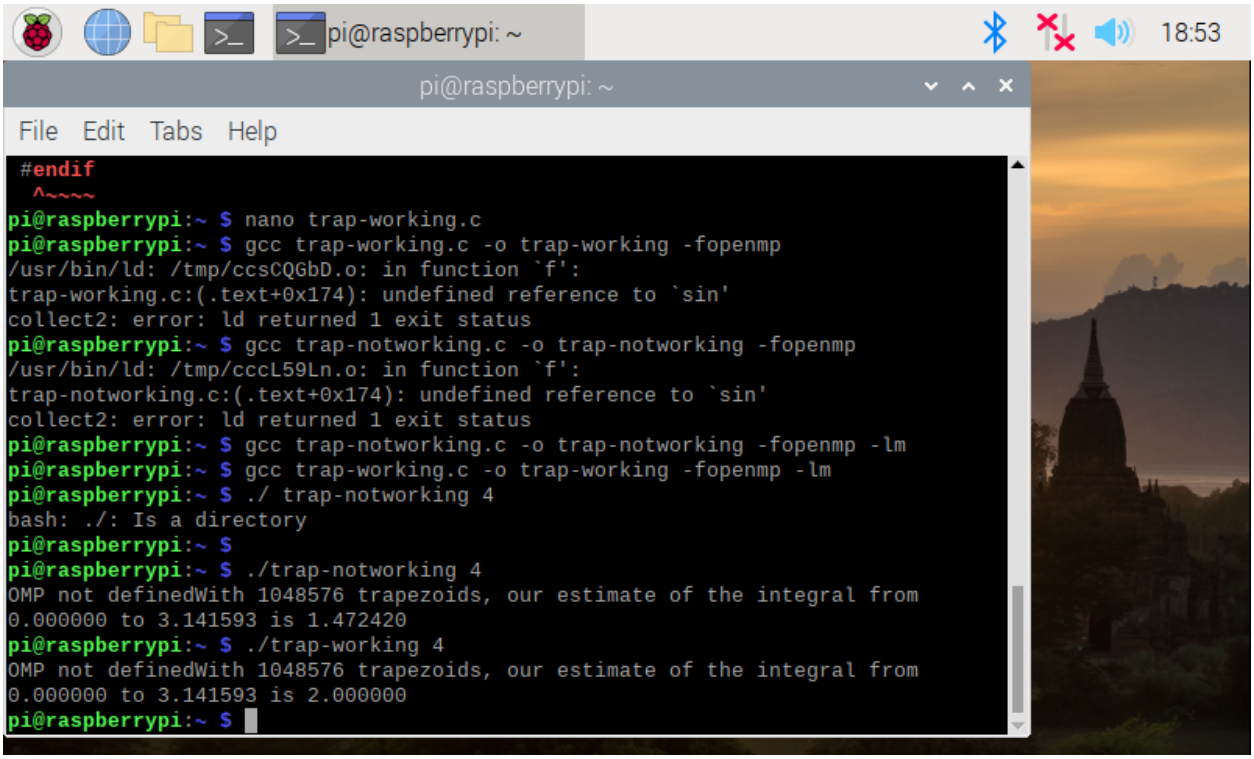
Both Statements are independent and can run parallel, the only dependent is [i] in the loop.

B) Parallel Programing Basics

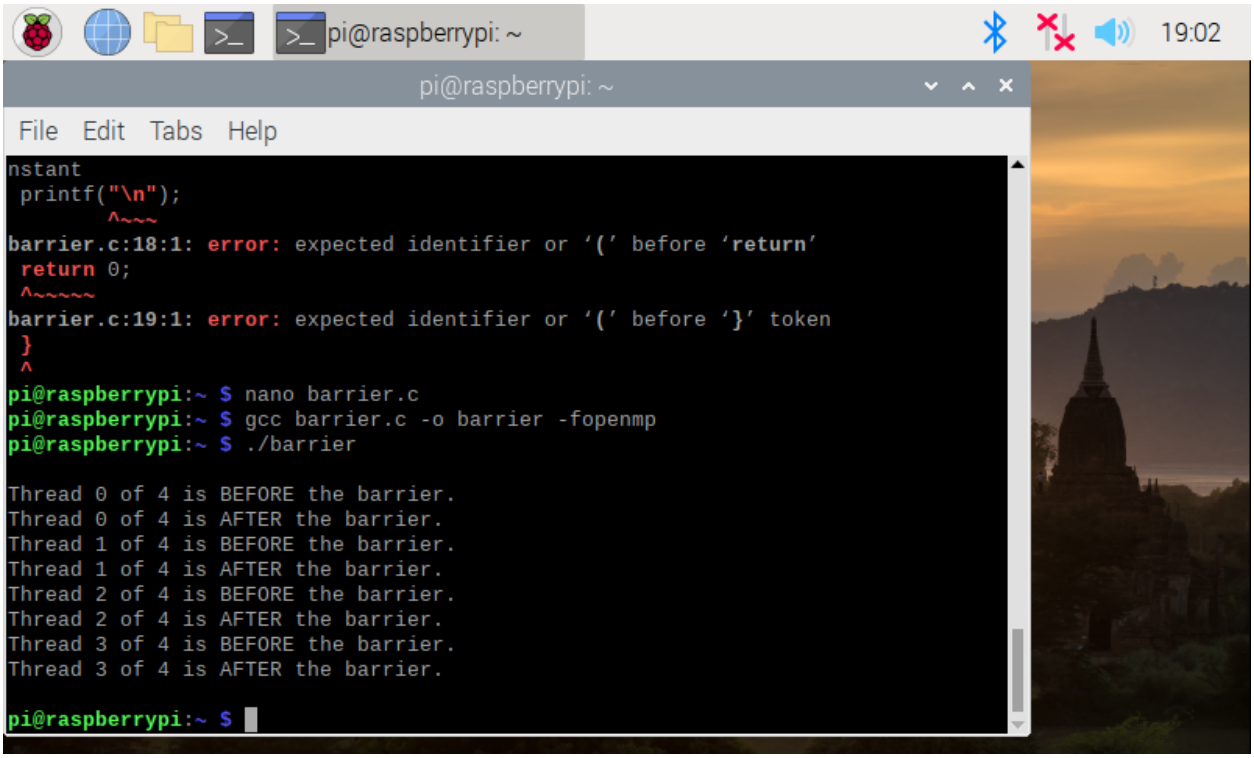
Task 4:

ARM Assembly Programming

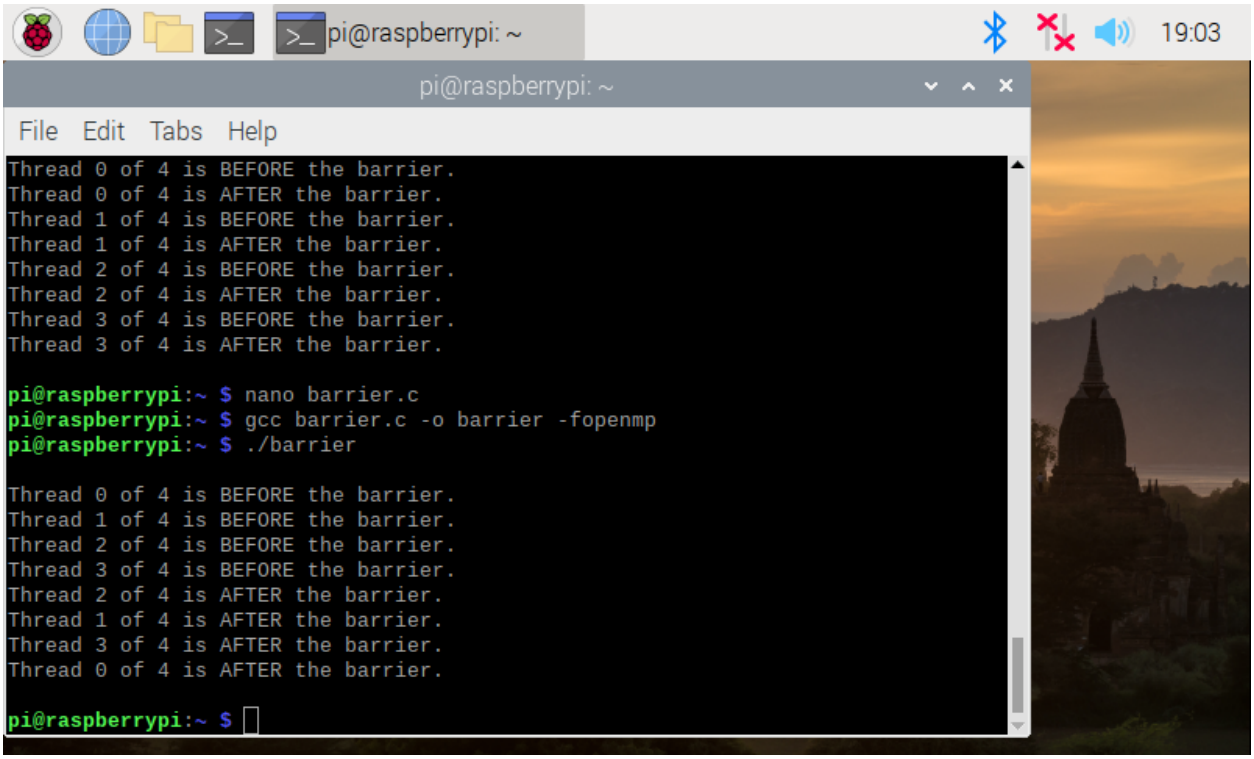
First copied and ran the program trap-notworking and trap-working to compare and observe the difference between the two.



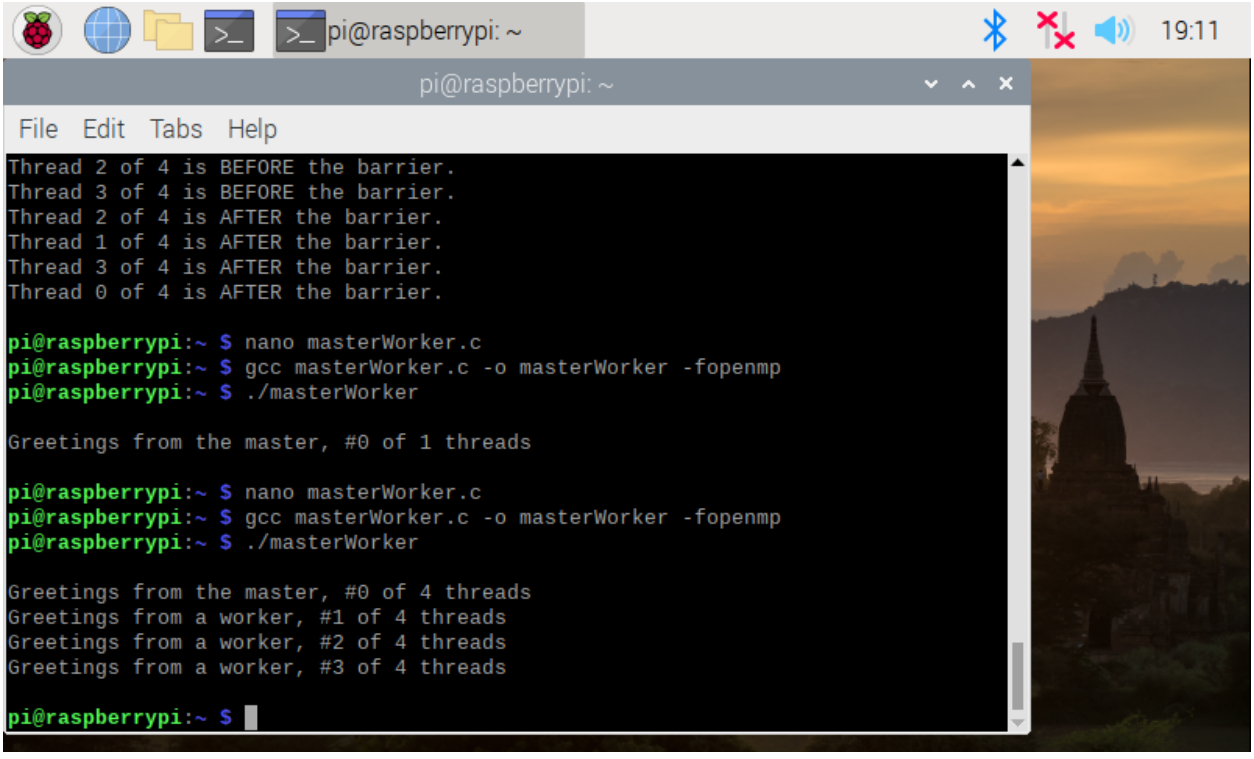
Next Copied the program Barrier and ran it.



Then compared with the pragma line at 31 being uncommented out.

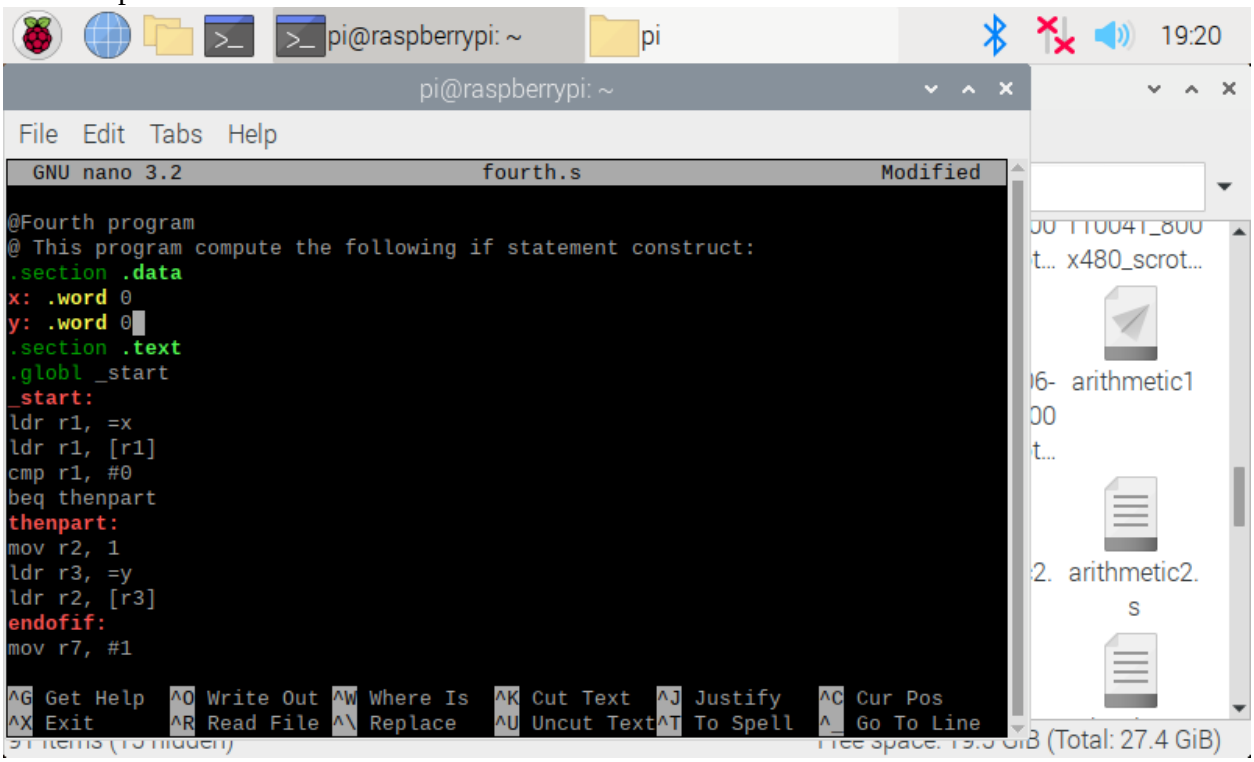


Next was the Masterworker program, the following picture is the compairison of the two programs running, the second excitucion is with the pragma line uncommented.

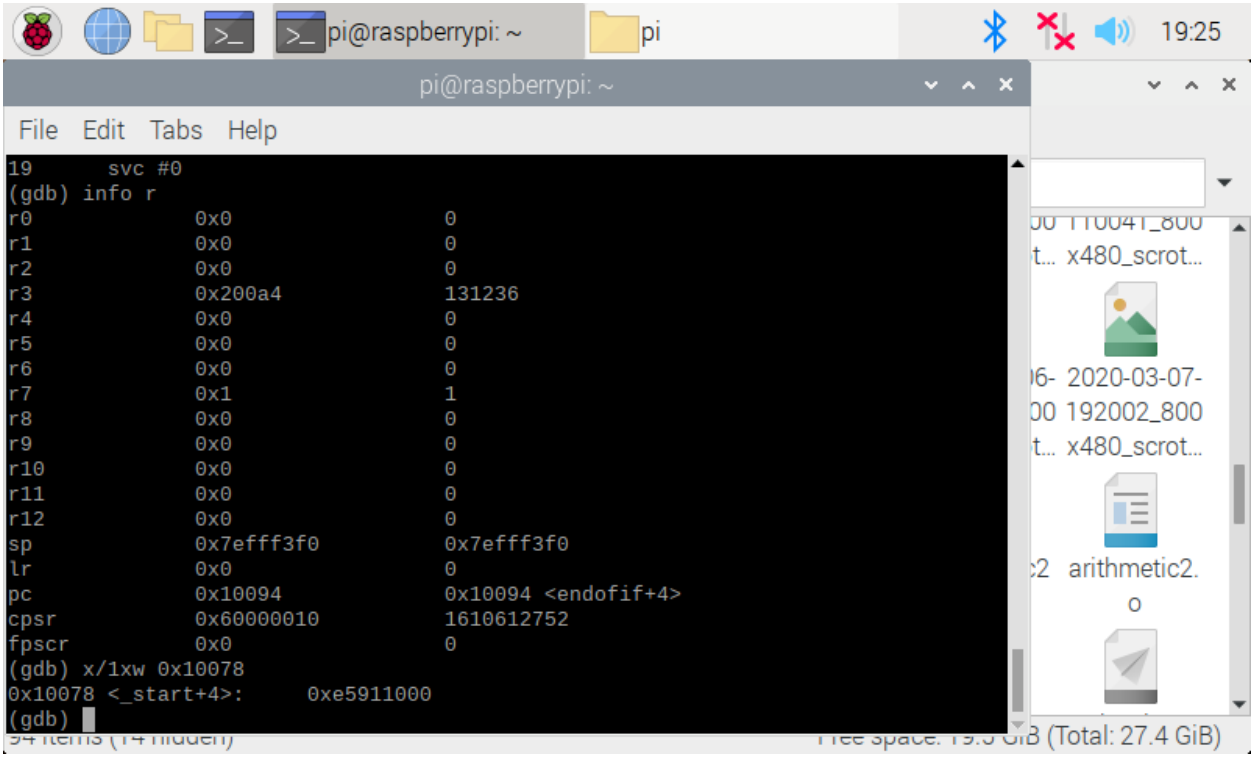


ARM Assembly Programming:

First I copied the code and saved as Fourth.s as instructed.



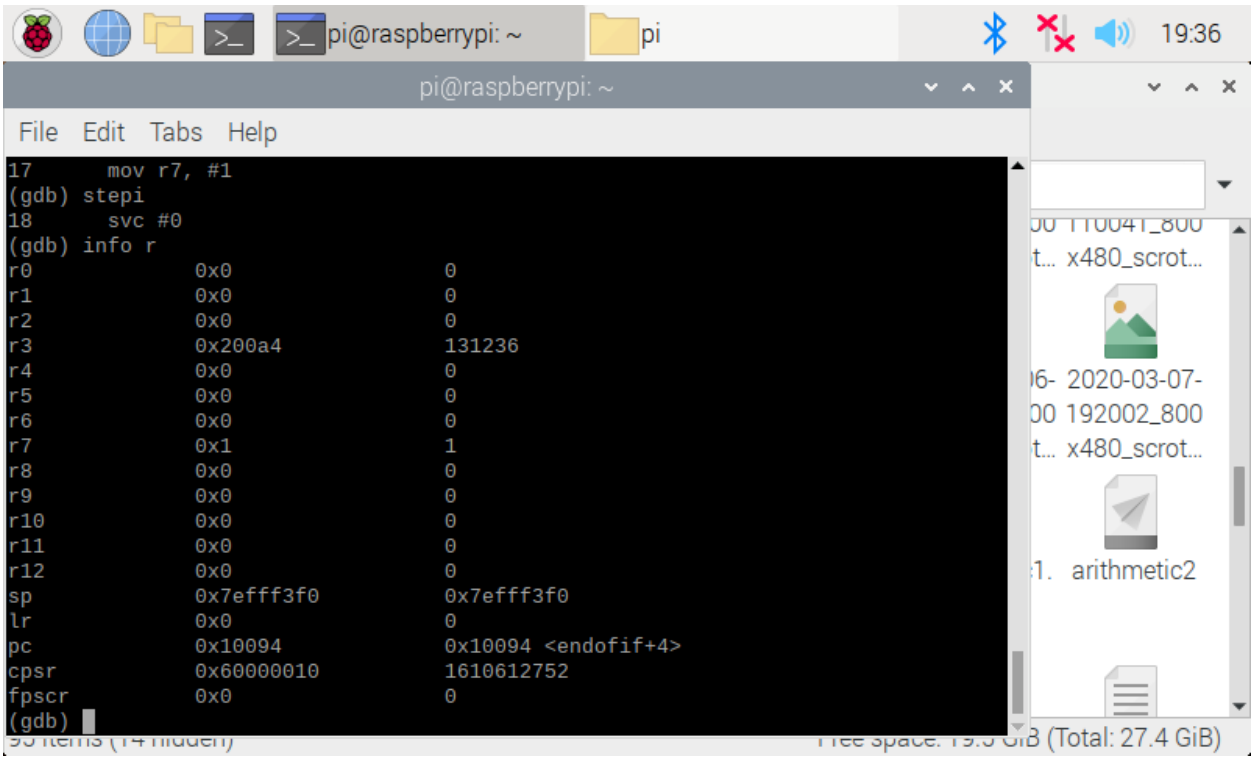
When into debug mode and at the end of the program I checked the value of the registers. We see that



We see that the negative flag is clear the zero flag is set the carry flag is set and the 2's complement flag is clear. In the CPSR

Part 2:

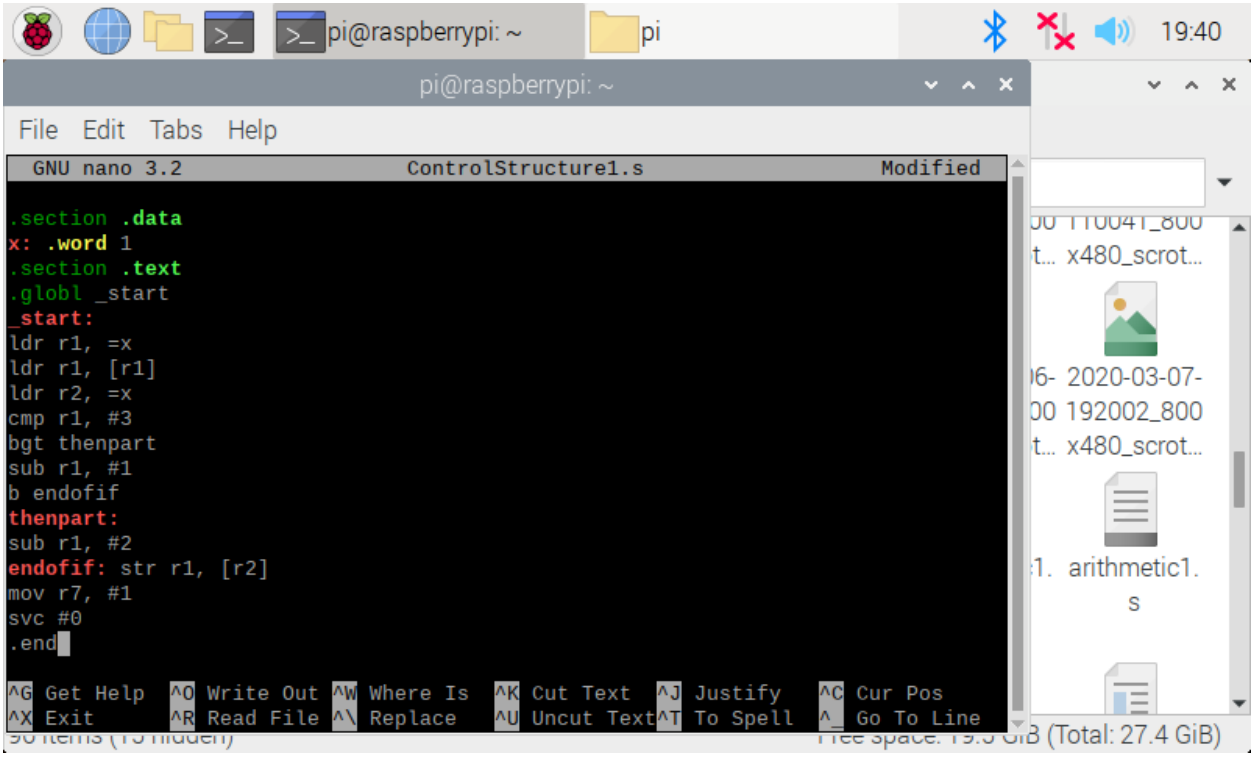
Using forth.s as a reference replaced the jump beq with bnq:

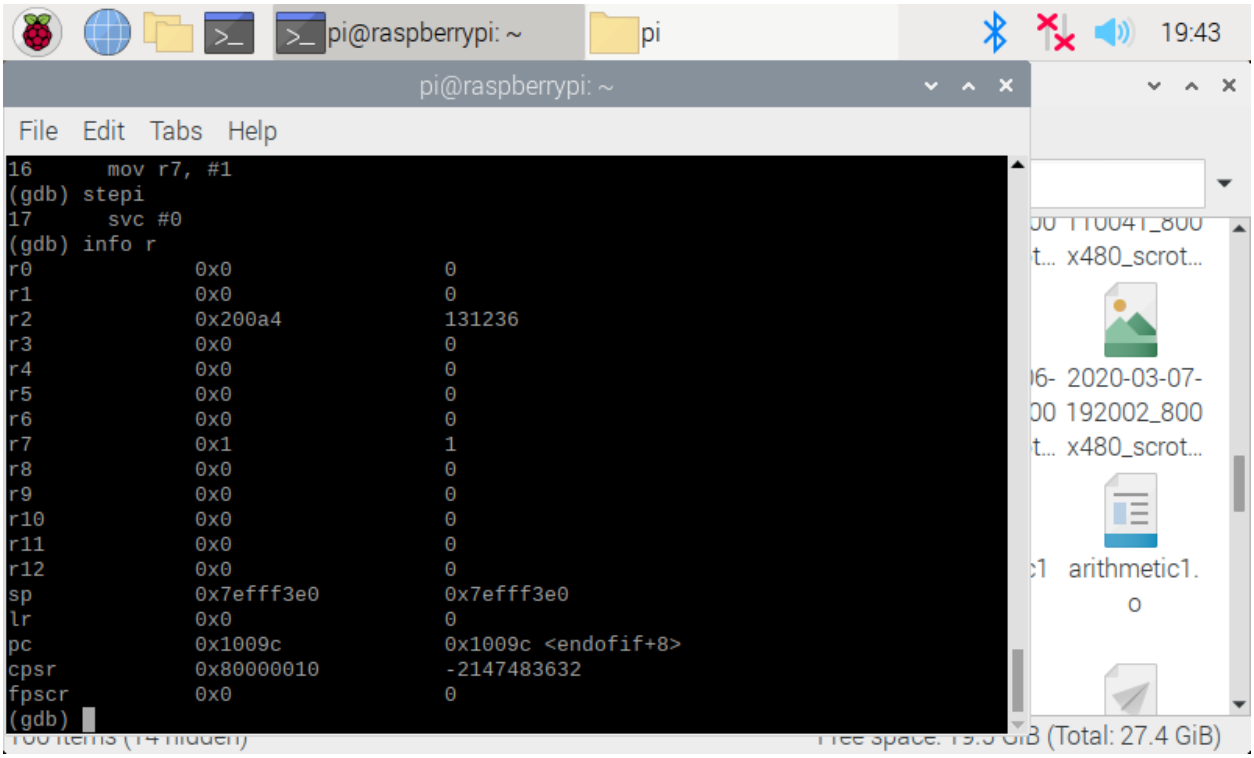


And noticed the status of the zero flag is cleared "0"

Part 3:

Using the fourth.s program as a reference again wrote a program to calculate the expression as instructed:





The value of the zero flag is clear “0” in this example.

Tek Acharya

Task 3a

a) (68p) Foundation

- (15p) Race condition:

(2p) What is race condition?

- ⇨ When a software, electronics, or other system depends on other event's outputs as its inputs, a hazard can occur if timing or sequence do not happen properly. This is bug as the programmer has not intended.

(5p) Why race condition is difficult to reproduce and debug?

- ⇨ Basically, race condition occurs in logic circuits, and multithreaded or distributed software programs. The race condition occurring in the logic circuit cannot be corrected through programming. And for bug due to relative timing between the intermediate threads in software, it is difficult to determine the end result as problems disappear in the production system when running in debug mode.

(8p) How can it be fixed? Provide an example from your Project_A3 (see spmd2.c)

The spmd2.c program wasn't working initially because, the race condition wasn't handled in the program, however, later when we declare the variable to be private, the race condition got fixed. We noticed that this time four core of the raspberry pi got separate threads to handle without creating hazard.

The following code has a problem because the race condition has not been handled.

```
#include <stdio.h>
#include <omp.h>
#include <stdlib.h
int main(int argc, char** argv) {
int id, numThreads;
printf("\n");
if (argc > 1) {
omp_set_num_threads( atoi(argv[1]) );
}
#pragma omp parallel {
id = omp_get_thread_num();
numThreads = omp_get_num_threads();
printf("Hello from thread %d of %d\n", id, numThreads);
}
printf("\n");
return 0;
```

Once we declared the local variables in the above code, we give private arguments to the cores and the problem got fixed.

```
#include <stdio.h>
#include <omp.h>
#include <stdlib.h
int main(int argc, char** argv) {
int id, numThreads;
printf("\n");
if (argc > 1) {
omp_set_num_threads( atoi(argv[1]) );
}
#pragma omp parallel {
int id = omp_get_thread_num();
int numThreads = omp_get_num_threads();
printf("Hello from thread %d of %d\n", id, numThreads);
}
printf("\n");
return 0;
```

- (15p) Summaries the Parallel Programming Patterns section in the “Introduction to Parallel Computing_3.pdf” (two pages) in your own words (one paragraph, no more than 150

words).

As the developers come along their practice of parallel programming, they categorized the pattern found into two main areas

1. **Strategies**
2. **Concurrent Execution Mechanism** (CEM)

As we write a program, we must consider a strategy that can cover the following points

1. What kind of **algorithmic strategy** to use?
2. Considering a strategy, what **implementation strategy** to apply.

The algorithmic strategy primarily deals with choosing the type of task that can be run concurrently by multiple processors. The parallel program often runs with several of this pattern at a time and contributes to an overall structure while some other deal with the data that is used to compute.

The CEM has two major categories namely

1. **Process/Thread Control Pattern** which heavily relies on how the process of parallel execution is controlled during runtime.
2. **Coordination Pattern** which conveys how these controls are coordinated during runtime.

- (12p) In the section “Categorizing Patterns” in the “Introduction to Parallel

Computing_3.pdf” compare the following:

o Collective synchronization (barrier) with Collective communication (reduction)

⇨

Collective Communication (reduction)	*Collective Synchronization (reduction)*
All processes reach to a specific location before executing.	All processes are blocked until a specific synchronization point is reached
One process of the communicator collects data from all other processes and perform an operation to find the result	It blocks the processes until all other processes are being reached to synchronization point successfully.
It uses MPI reduce() function	It uses MPI barrier() function
This a type of parallel application of computing	This is a type of parallel application of computing
This uses concurrent execution mechanism for parallel execution	This uses concurrent execution mechanism for parallel execution
This is a type of coordination system of parallel computation	

o Master-worker with fork join

Master-worker	Fork join
Main process is being divided into small chunks which in turn being distributed to several worker processes.	Pattern which is used to execute parallel light weight processes and threads.
The master handles one of the threads while rest is handled by the workers	Threads from a sequential mode is forked into several slave threads and are joined back once tasks for each is completed resulting into joined output.

- (26p) Dependency: Using your own words and explanation, answer the following:

(3p) Where can we find parallelism in programming?

⇨ Parallelism is found almost every modern computational machine. As parallelism yields the output faster yet accurate as in mobiles, server machines, virtual reality, database etc.

(6p) What is dependency and what are its types (provide one example for each)?

⇨ When an input of one execution depends on the output or resource of another, such a scenario is called dependency in computing.

⇨ There are three main categories

a) True Dependence

S1: x = 1

S2: y = x This can be avoided by directly passing a value of x to y i.e. y = 1

Here, the value of second is dependent on the value of first.

b) Output Dependence
S1: x = f(a)
S2: y = x This cannot be avoided as the value of x is to be evaluated before any Value of y can be assigned
Here, the value of second is dependent on first

c) Anti-Dependence
S1: x =y
S2: y =1 This cannot be avoided as the value of first can only be accessed only once
Second is accessed.
Here, the value of first variable is dependent on the value of second variable

(3p) When a statement is dependent and when it is independent (Provide two examples)?

⇨ When the execution of
A. statement1;
statement2;
B. statement2;
statement1; if pattern A and B yields same result
then the two statement1 and statement2 are independent of each other. And if the they yield different result; they are dependent on each other.
For example,
S1: x = 1 | S2: y = 2
S2: y = x | S1: x =1
In the above example if the statement S1 and S2 are handled by two cores separately, the order does not matter and has the same output.
But if the same statements are handled by a single core, the order matters and the dependency occur.

(3p) When can two statements be executed in parallel?

⇨ If there are no dependency between two or more statements in a cycle, then the parallel execution can be applied.

(3p) How can dependency be removed?

⇨ Some dependencies can be resolved by rearranging and eliminating the statements before execution.

(8p) How do we compute dependency for the following two loops and what type/s of dependency?

```
for (i=0; i<100; i++)
   S1: a[i] = i;
```

```
for (i=0; i<100; i++) {
   S1: a[i] = i;
   S2: b[i] = 2*i;
}
```

We compare the statement sets of IN and OUT at each node to see the data dependence relations.

The statement S in a set of IN and OUT is defined as follows:

IN(S) = the S uses the set of memory location.

OUT(S) = The S modifies the set of memory location.

We use two basic strategies to find the dependencies of a statement, these include

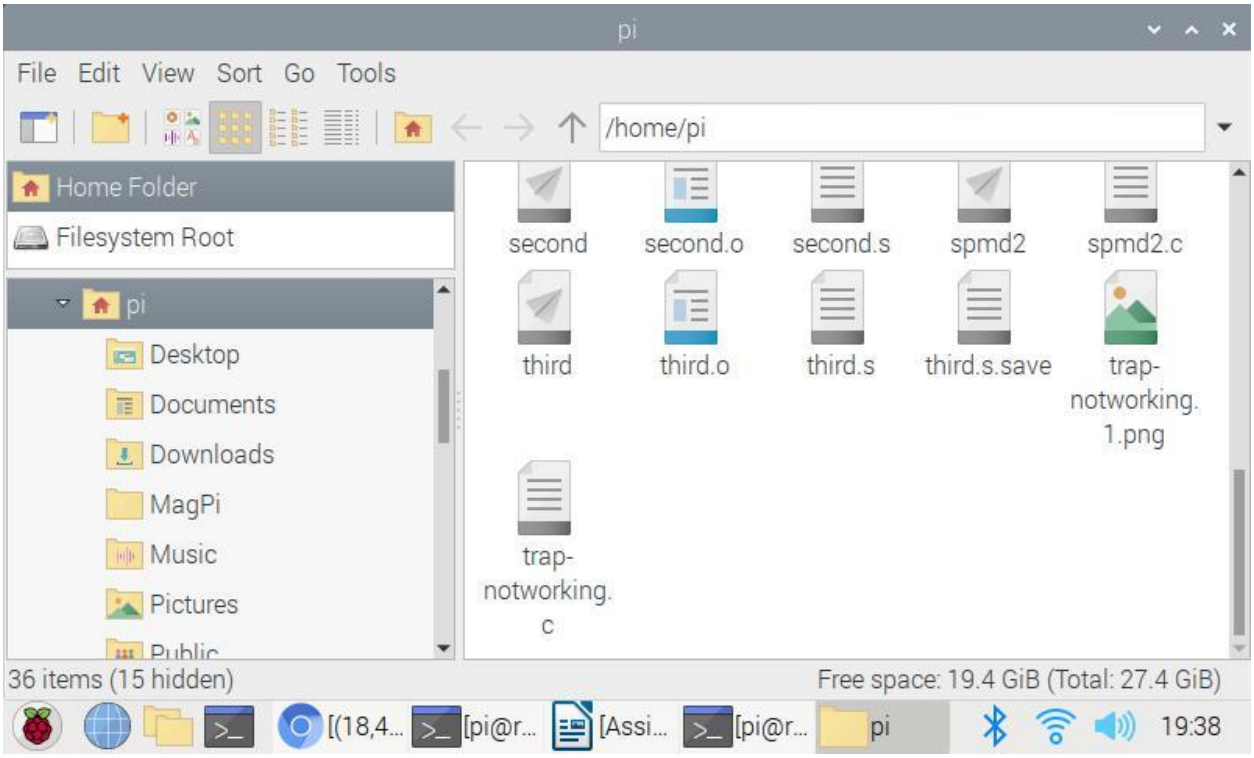
1. Unrolling of the loop (if any) into separate statements or iterations
2. Reveal or study the relationship between the statements

In our first loop above, S1(1), S1(2), ……, S1(99) are independent of each other.

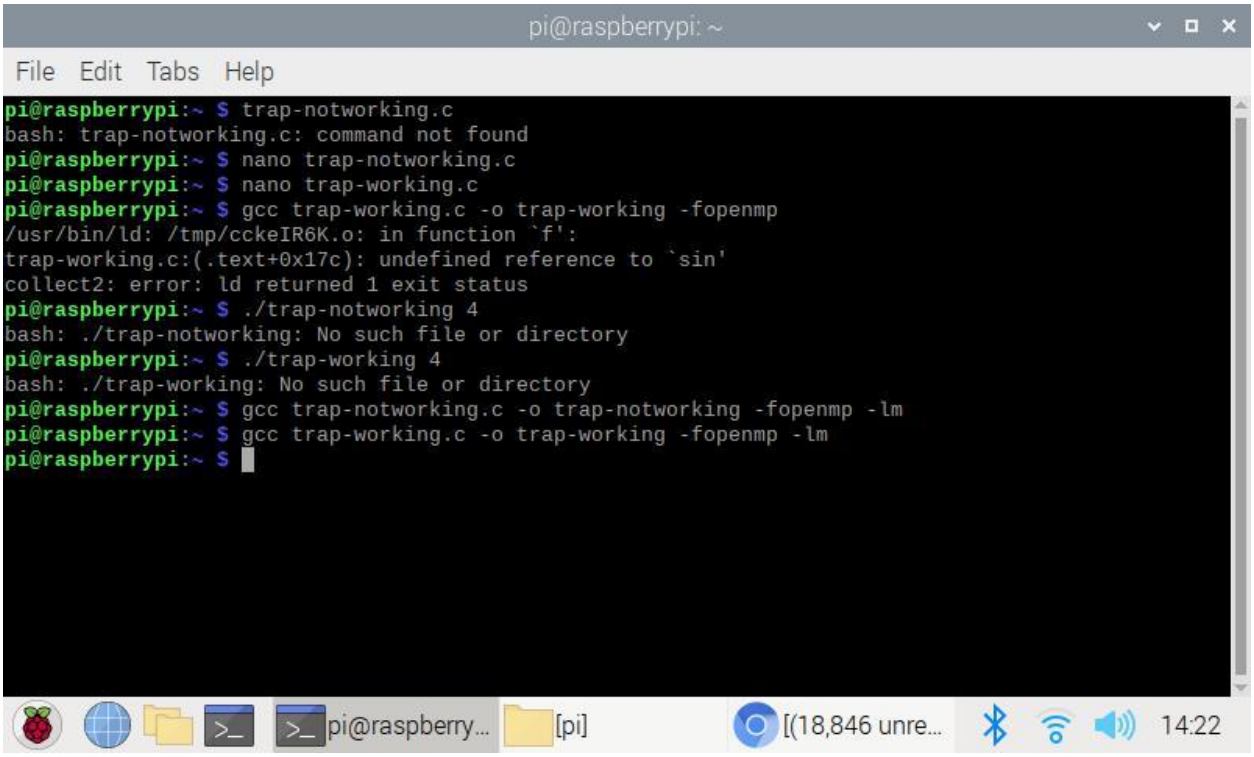
Whereas, in the second loop, S1(1), S1(2), …., S1(99) are independent of each other but statement S2(1), S2(2), …., S2(99) depends on S1 as we can see this after unrolling the above statements.

TASK 3b

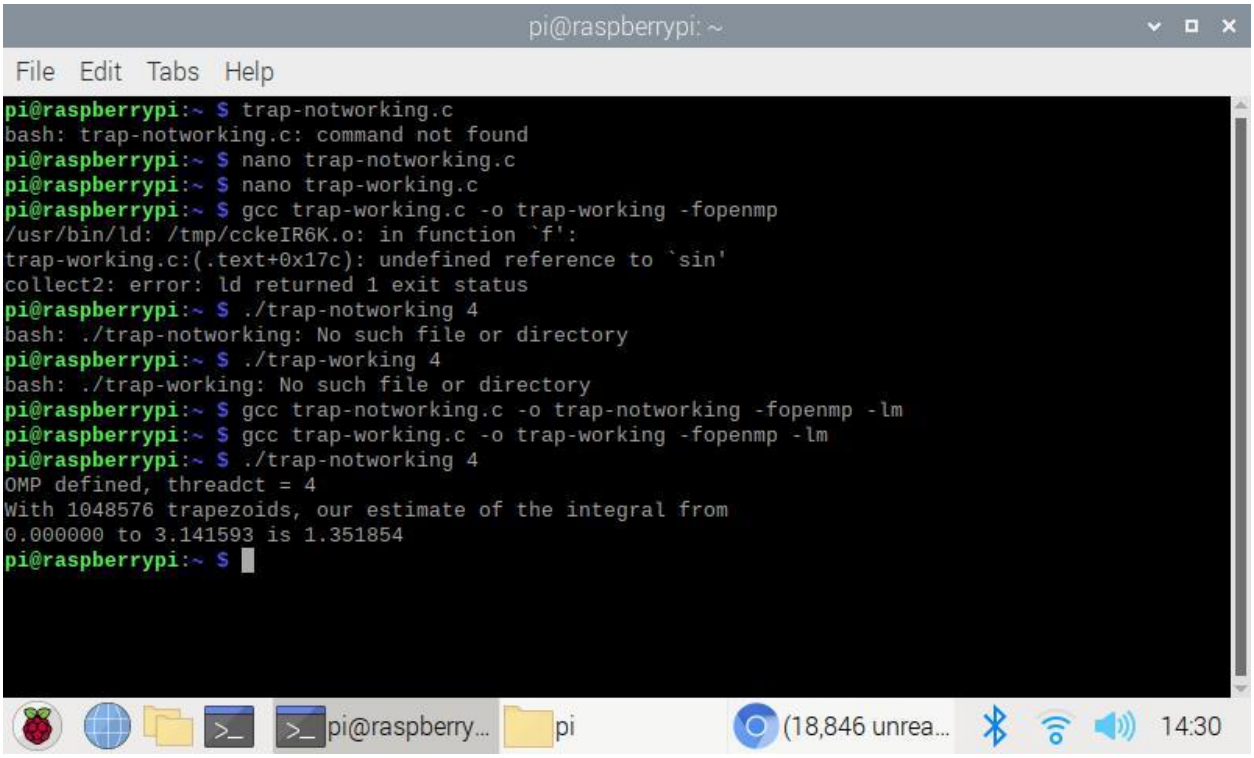
Copied and pasted the code, and created an executable as bellow



The executable created shows that there was no programming error.

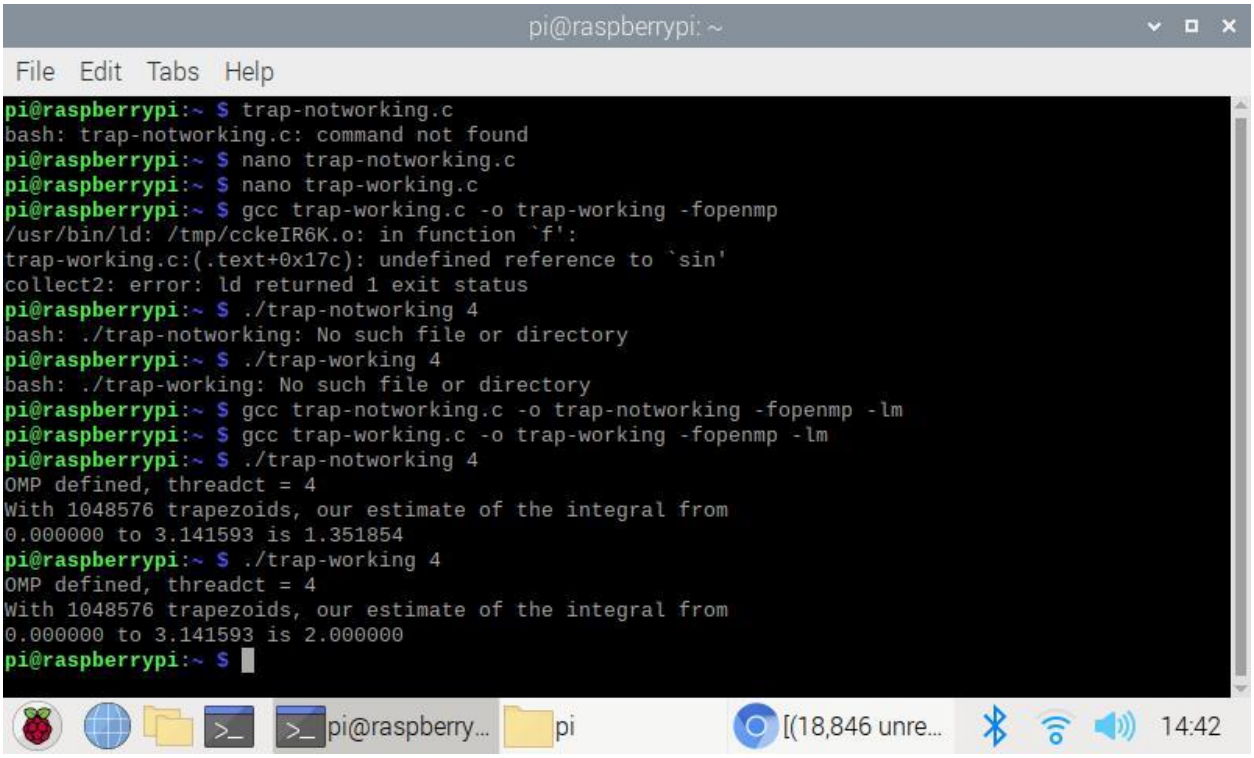


Trap-notworking successfully compiled and ran



The result is not correct as we can see from the above value of our integral (2).

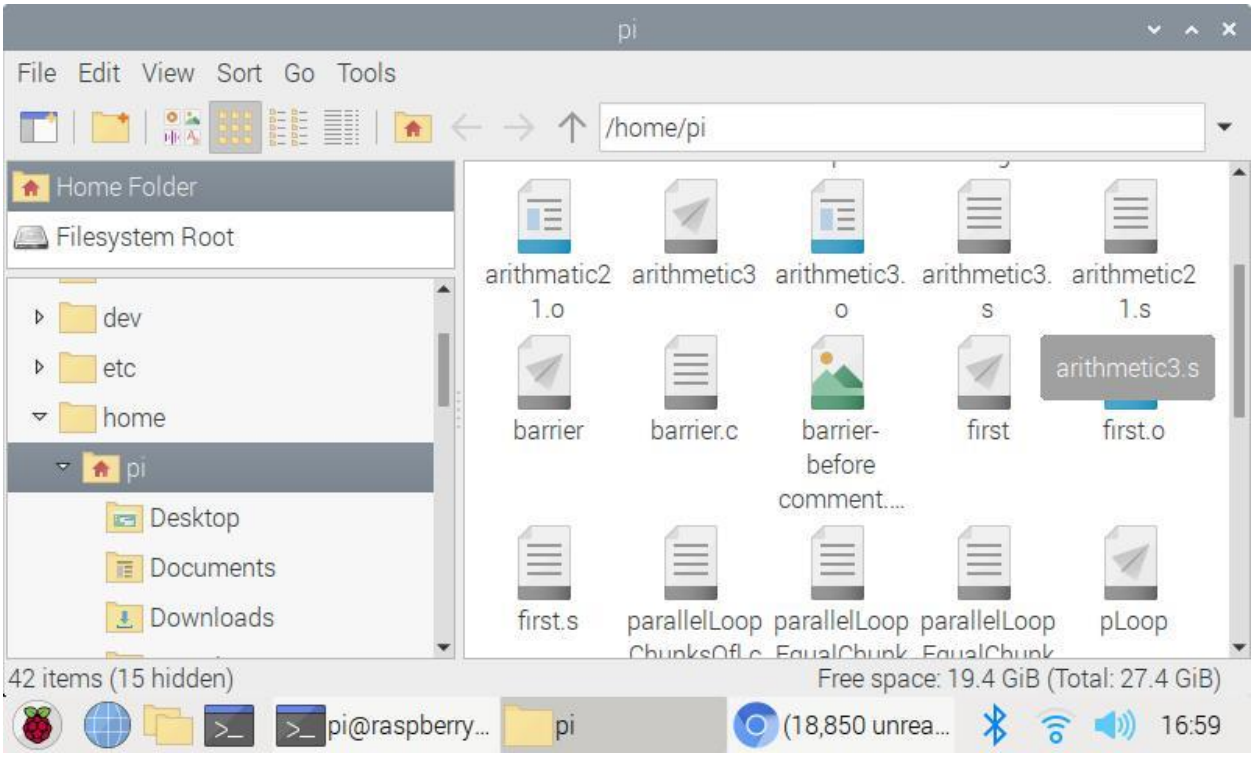
Then, compiled and ran with trap-working version and got the correct expected value of our integral as shown below.



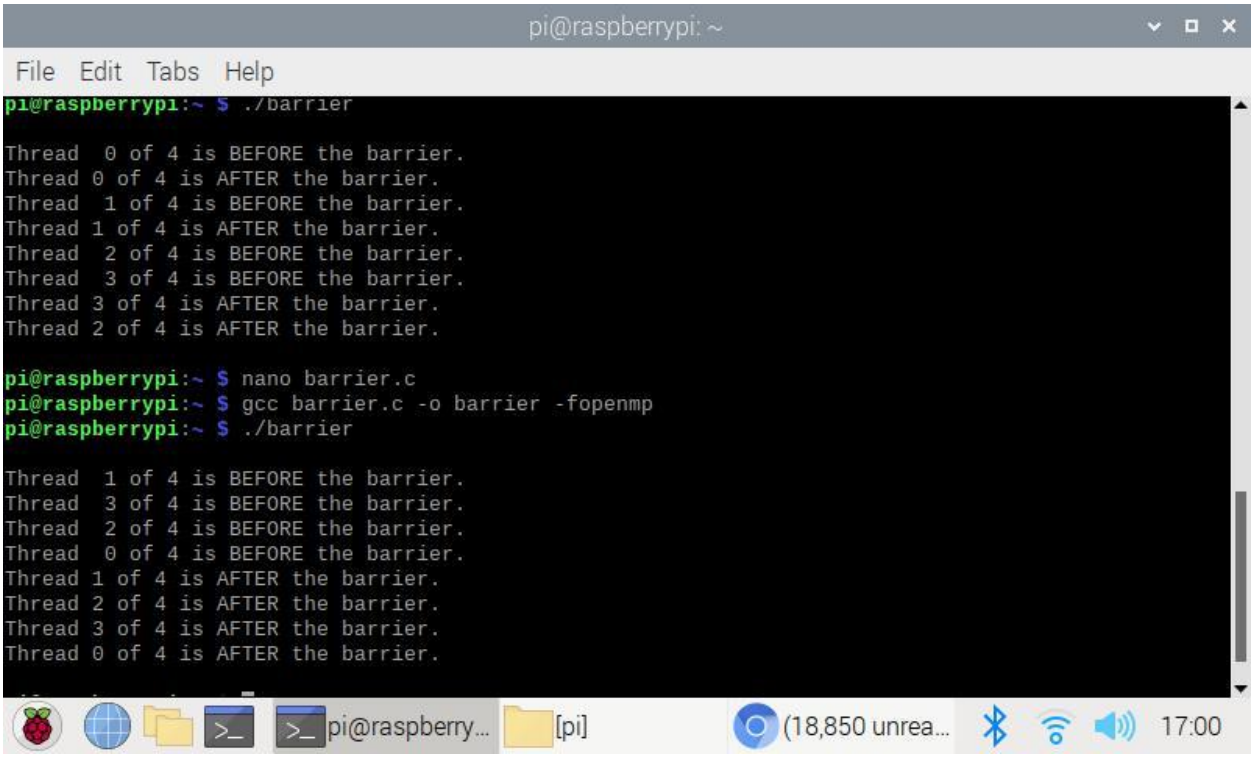
The reason behind the trap-notworking version giving wrong answer is due to race condition. The program is trying to read a variable that is changed by different threads. Using \ at the end of #pragma will save the last iteration data for use outside the #pragma clause. Here, the threads are not independent of each other.

Barrier

(without barrier condition)

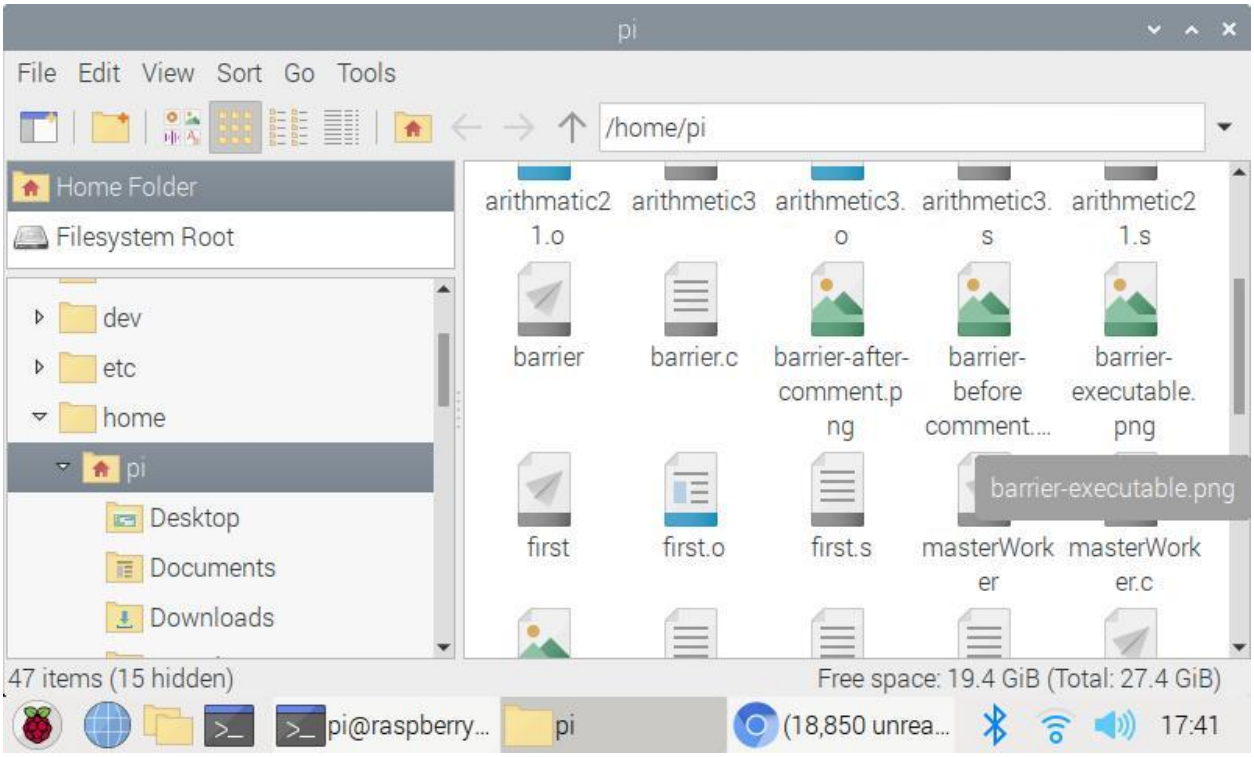


Executable created for barrier.c



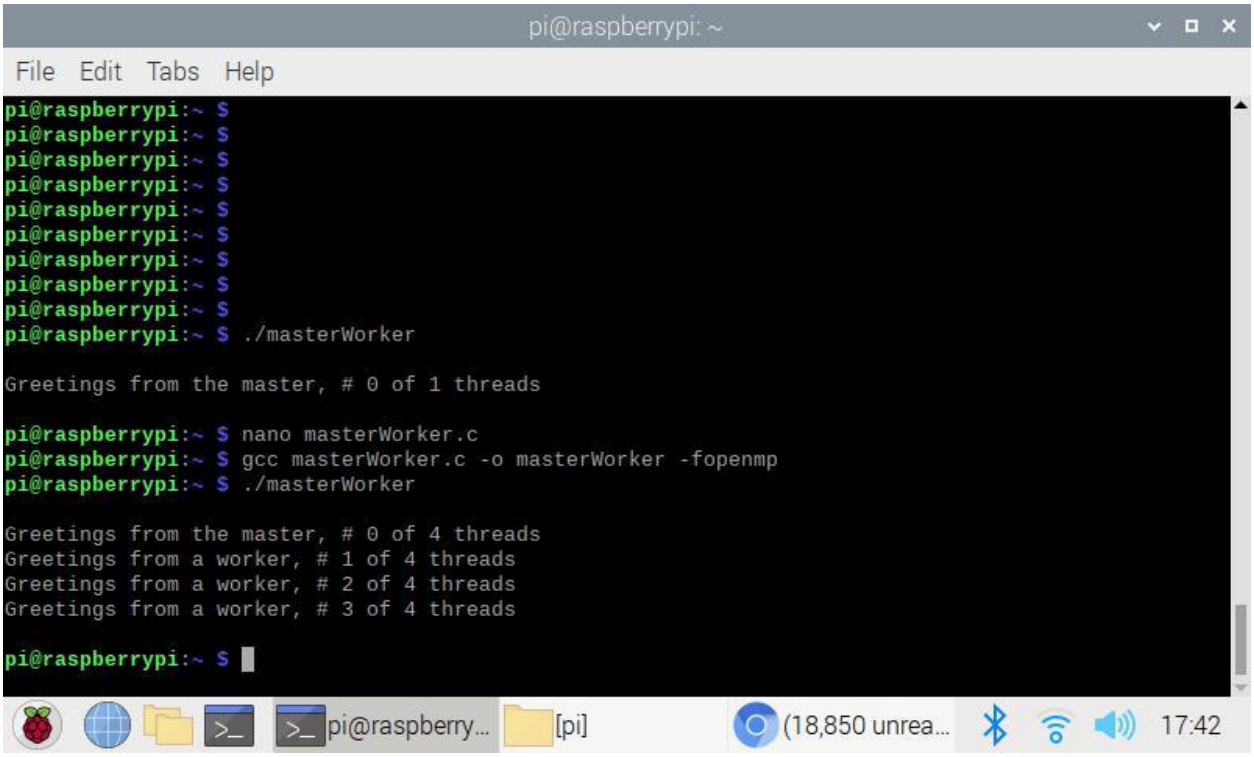
The difference between with and without barrier is that the one without barrier, each thread completes their tasks whereas with the barrier, all threads recombined at a point and complete their tasks only upon all tasks has been completed by each thread. Here, threads will wait until all threads meet at a point and then proceeds further execution.

MasterWorker



masterWorker executable created

Worker-master without pragma and with pragma



In the one with commented pragmas, no threads are being used, so just sequential computation.

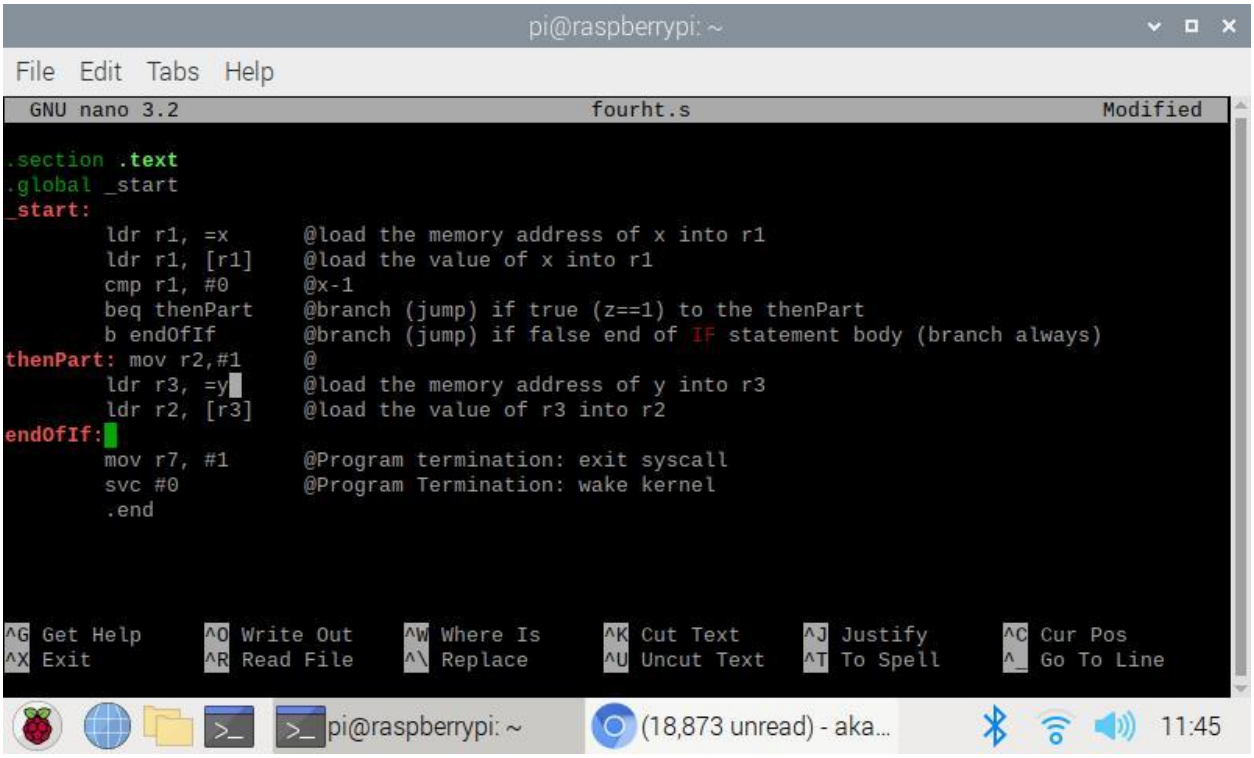
But once the comment is lifted, the threads has been activated to four different worker-threads and one still handled by the master thread.

This is an example of masterWorker computation.

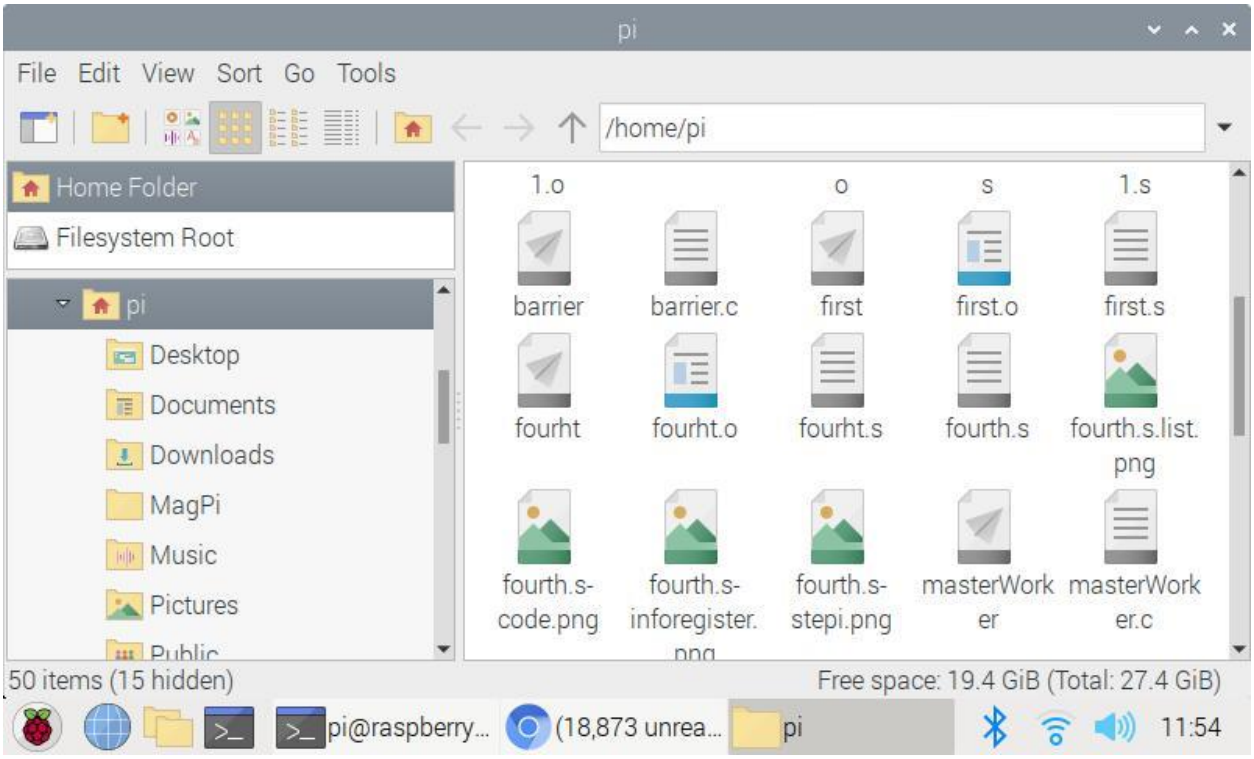
Task 4a

Part1

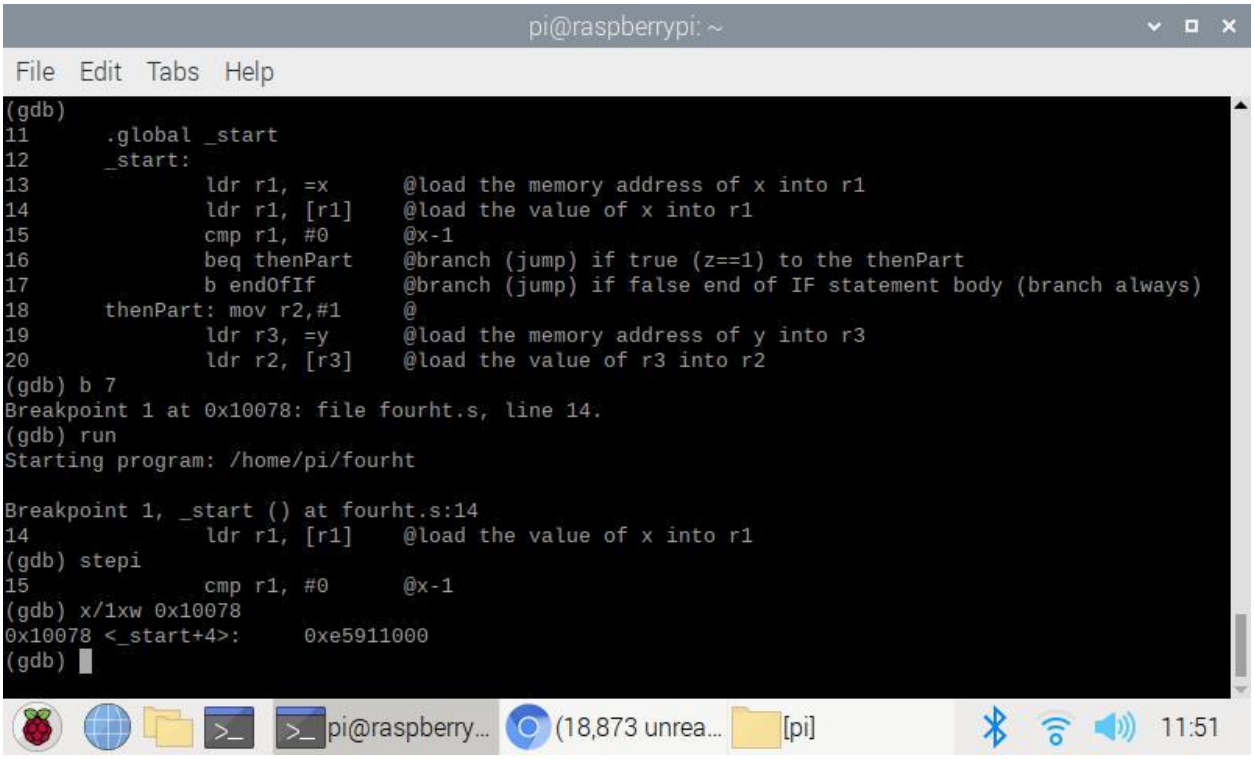
Note: By mistake I typed fourth.s to be fourht.s. Pardon me



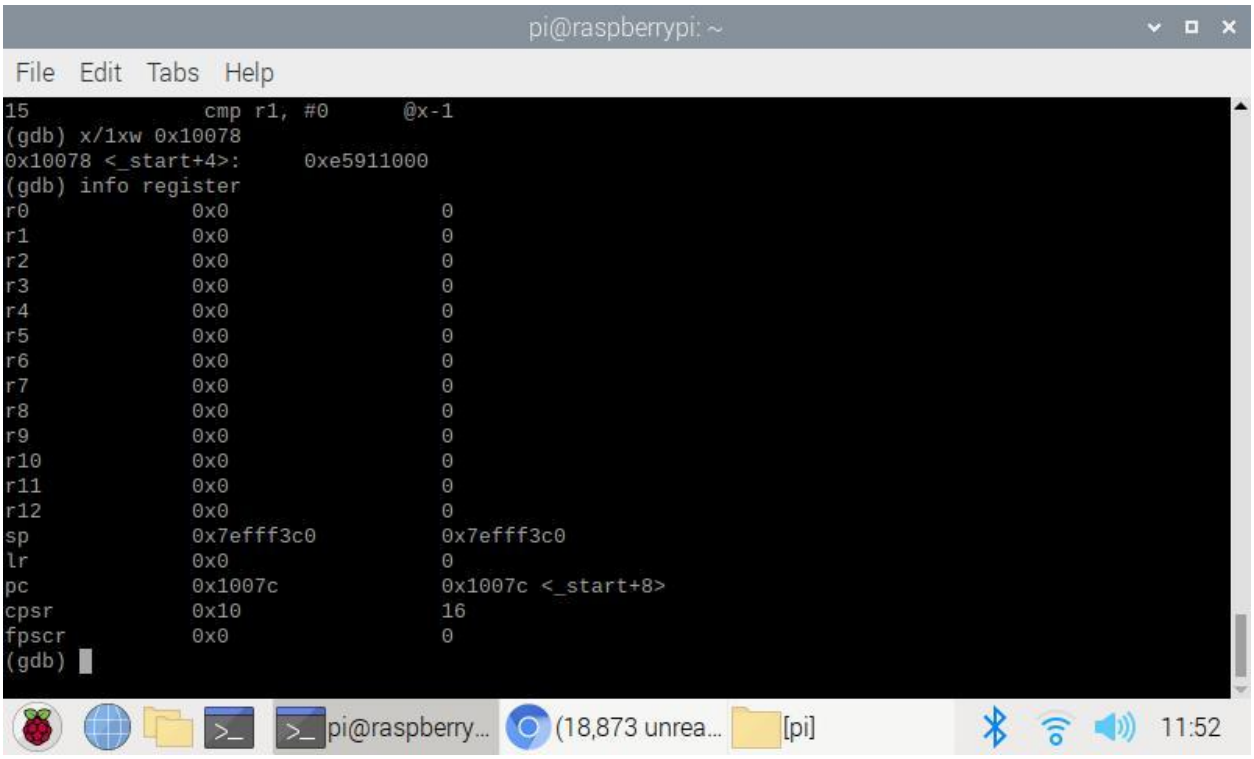
fourth.s copied and pasted



Executable created



Compiled, linked, used GDB to debug. Breakpoint set at line 7 but it jumped to line 14. I then step in and checked the value of memory and register as shown below. I also examined the code part where the main logic is used (cmp) and monitored how the jump occurred using debugger.

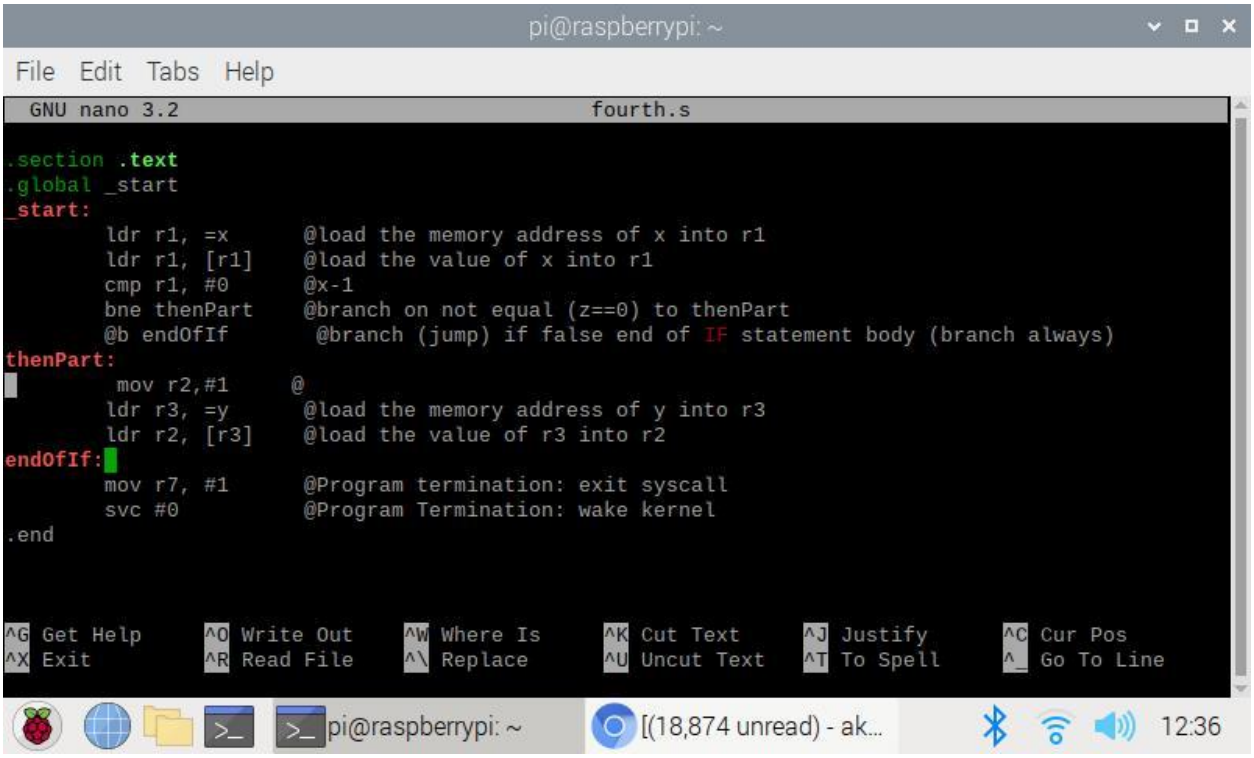


(the end result of the execution)

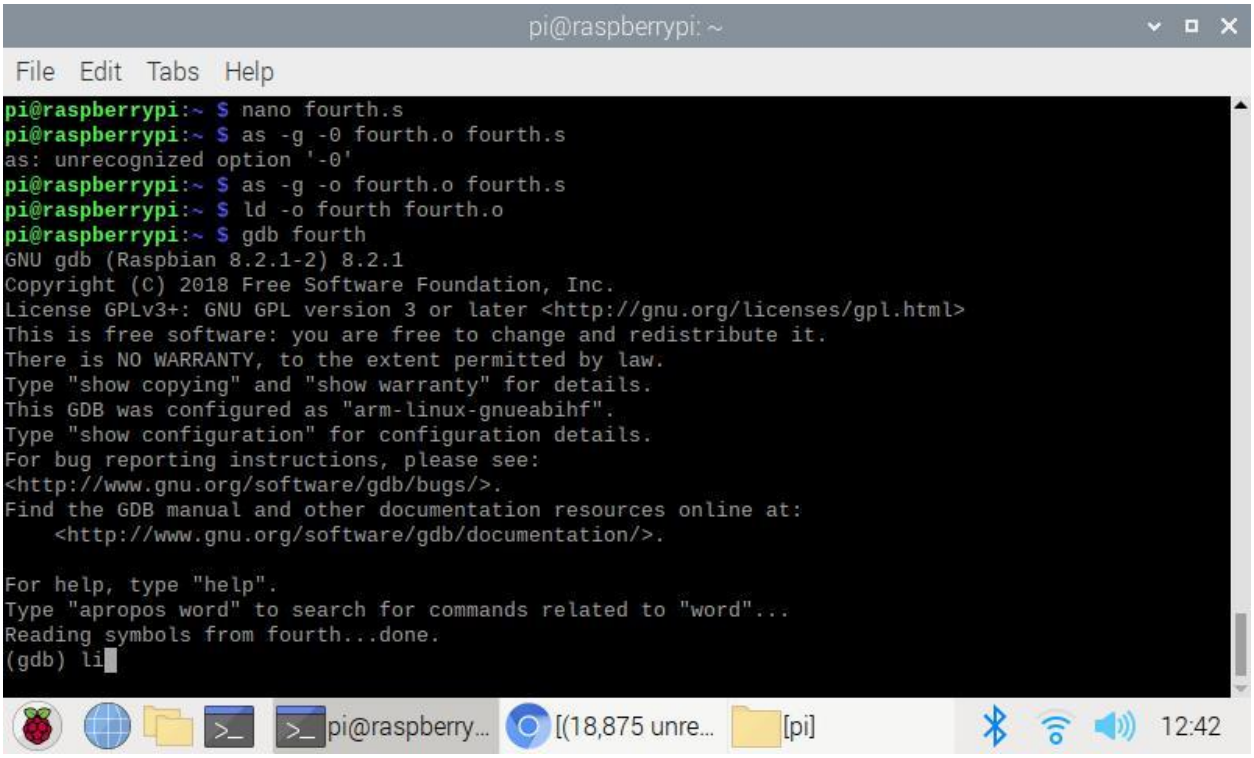
While using stepi and at point 0x10078, the hex value of 0xe5911000 is the data that is being stored at. At this time the r3 value address is 0x200a8.

While looking into the value of cpsr = 10 means the z (zero flag) value is set which will make jump to thenPart and thus the final value of r2 is 0. Also, while stepping in the then value of the flags I noticed the number 6000010 of cpsr which represents the 0 1 1 0 negative, carry, and 2's complement, OV respectively.

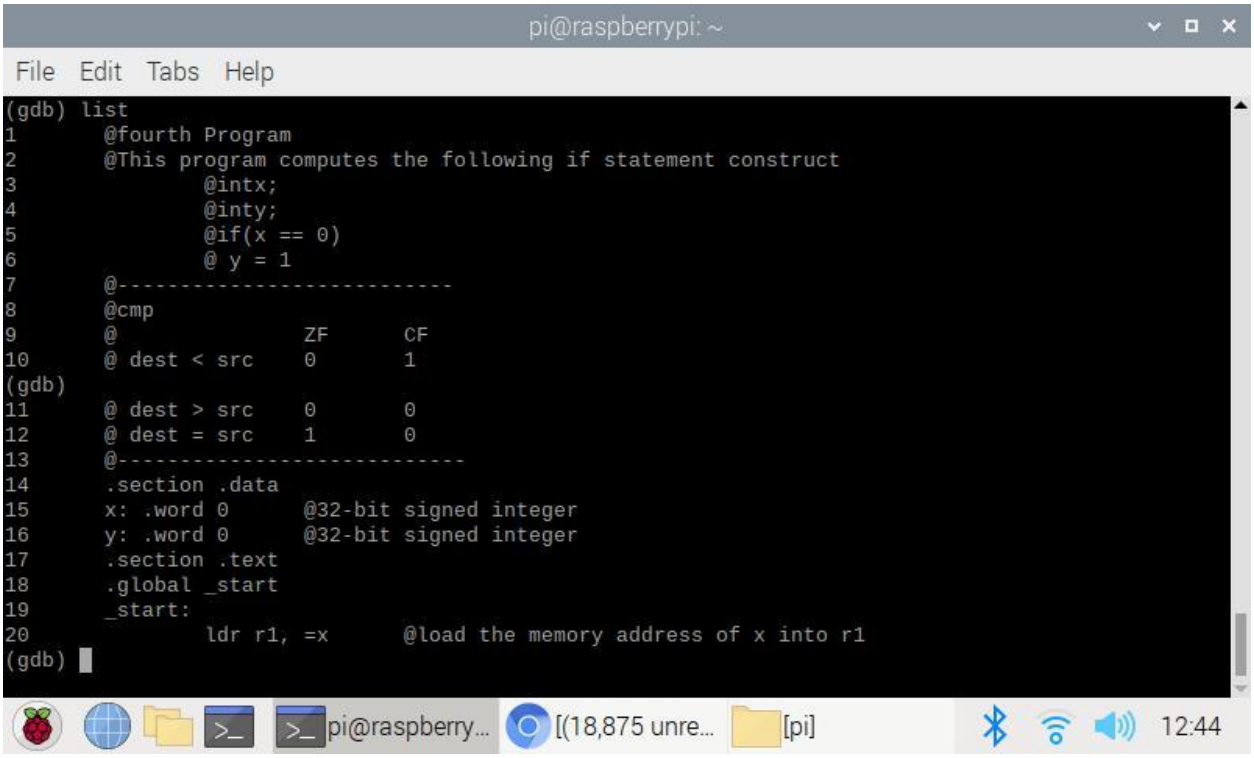
Prt2. Improved Code for fourth.s



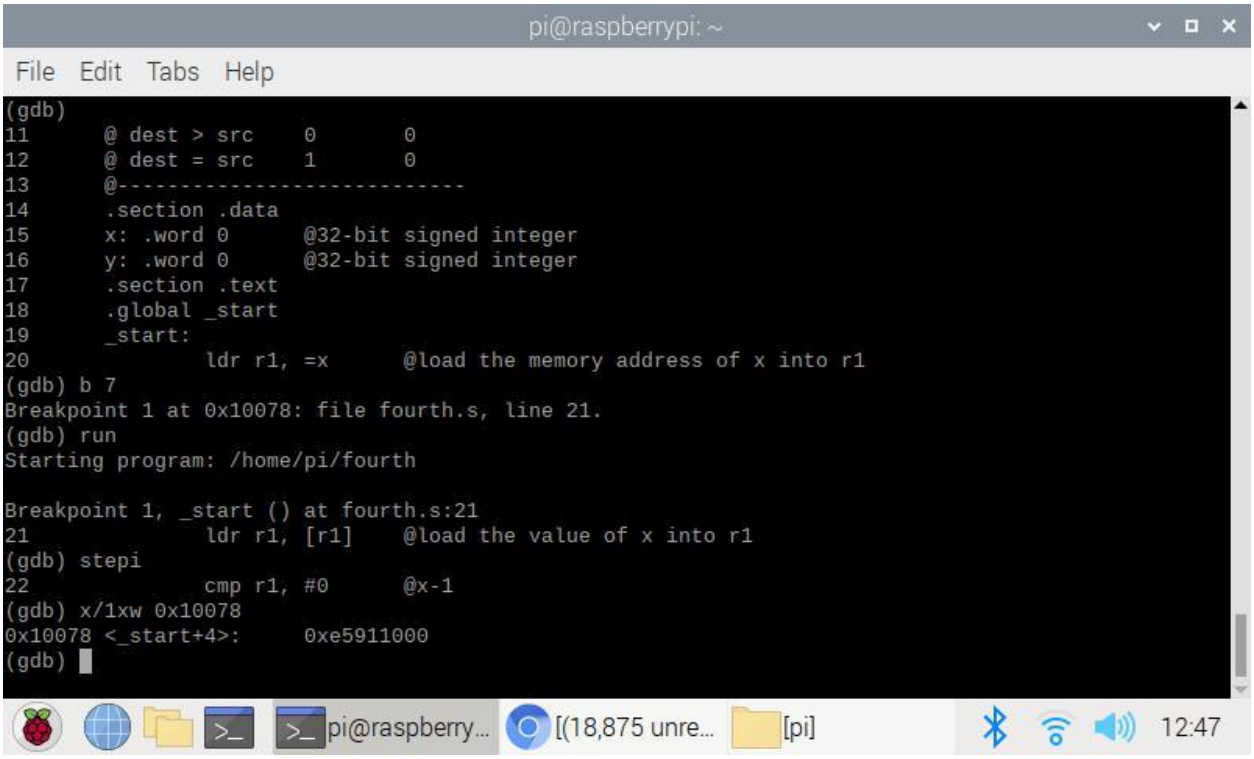
Corrected the code for its better look and efficient avoiding back-to-back branches. In this code, we removed ‘b’ instruction and used ‘bne’ instead that is the refixation of ‘beq’ using De-Morgan’s law.



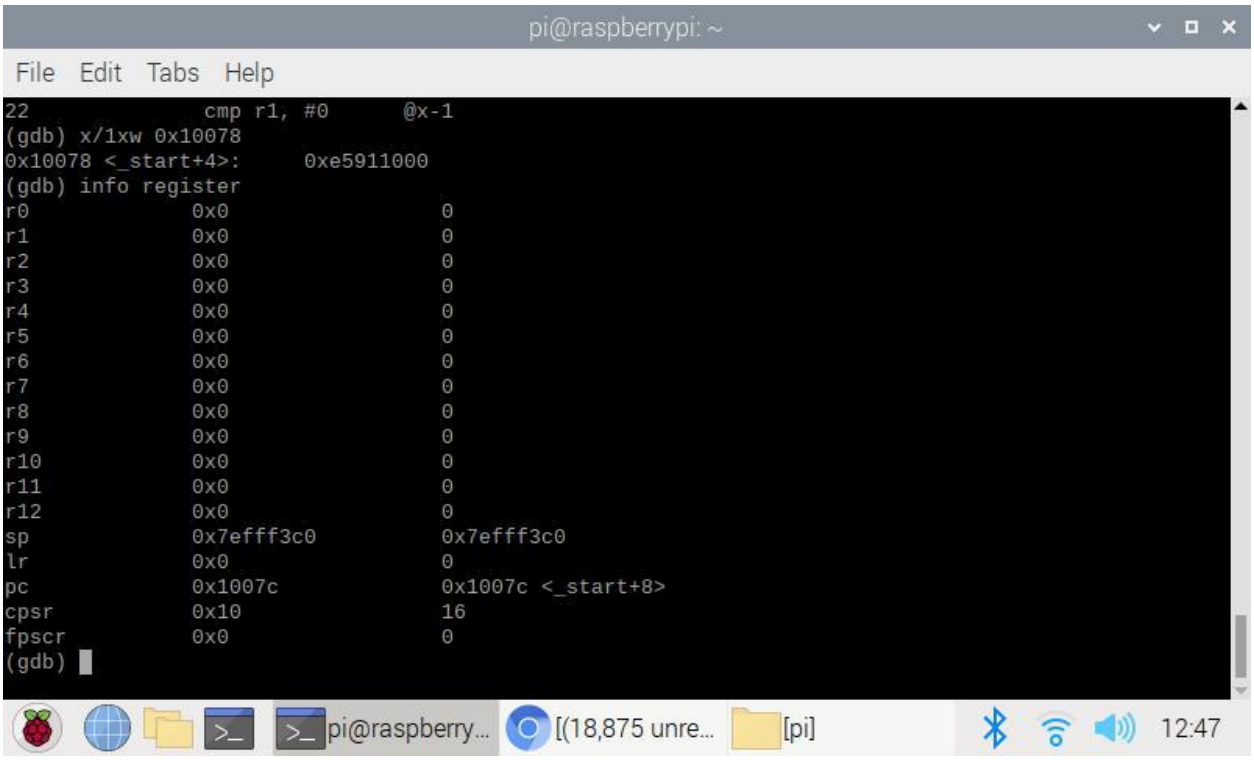
Compiled and linked



Looked into list view for better debugging.



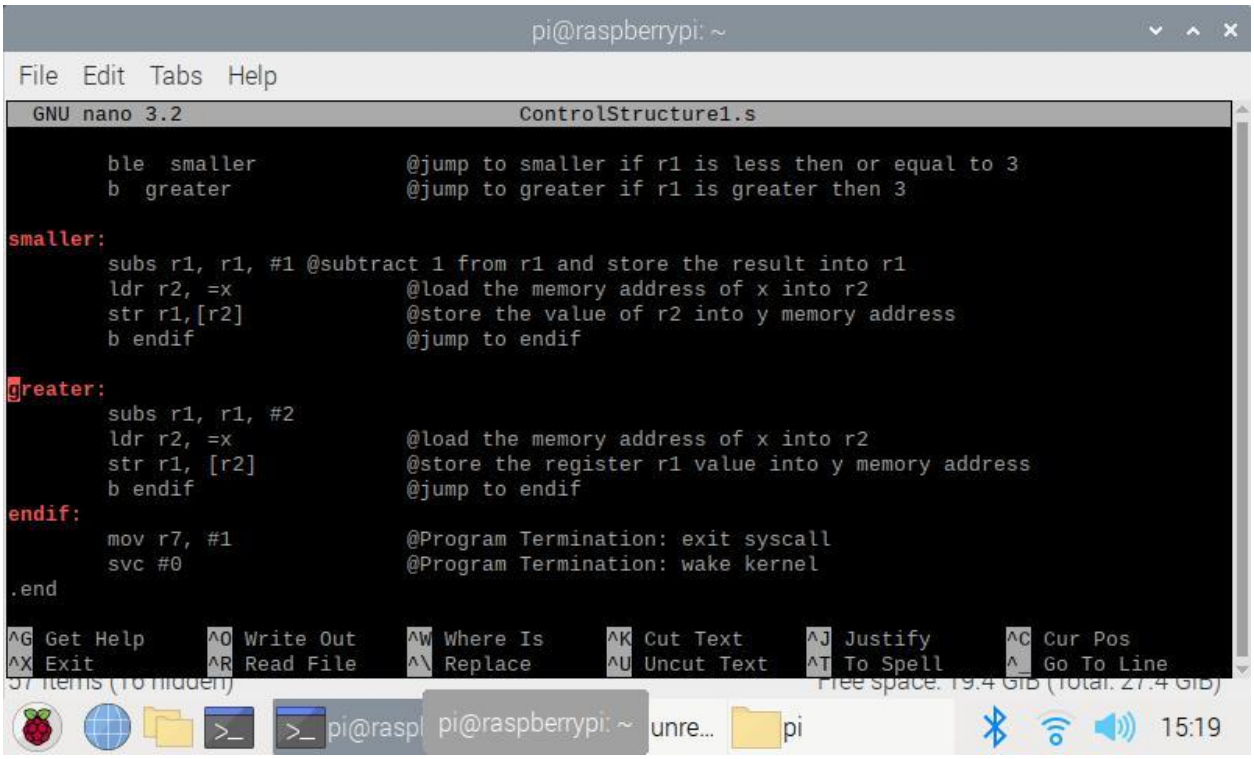
Debugged starting line 21 and stepped in into each line and examined the memory and register content.



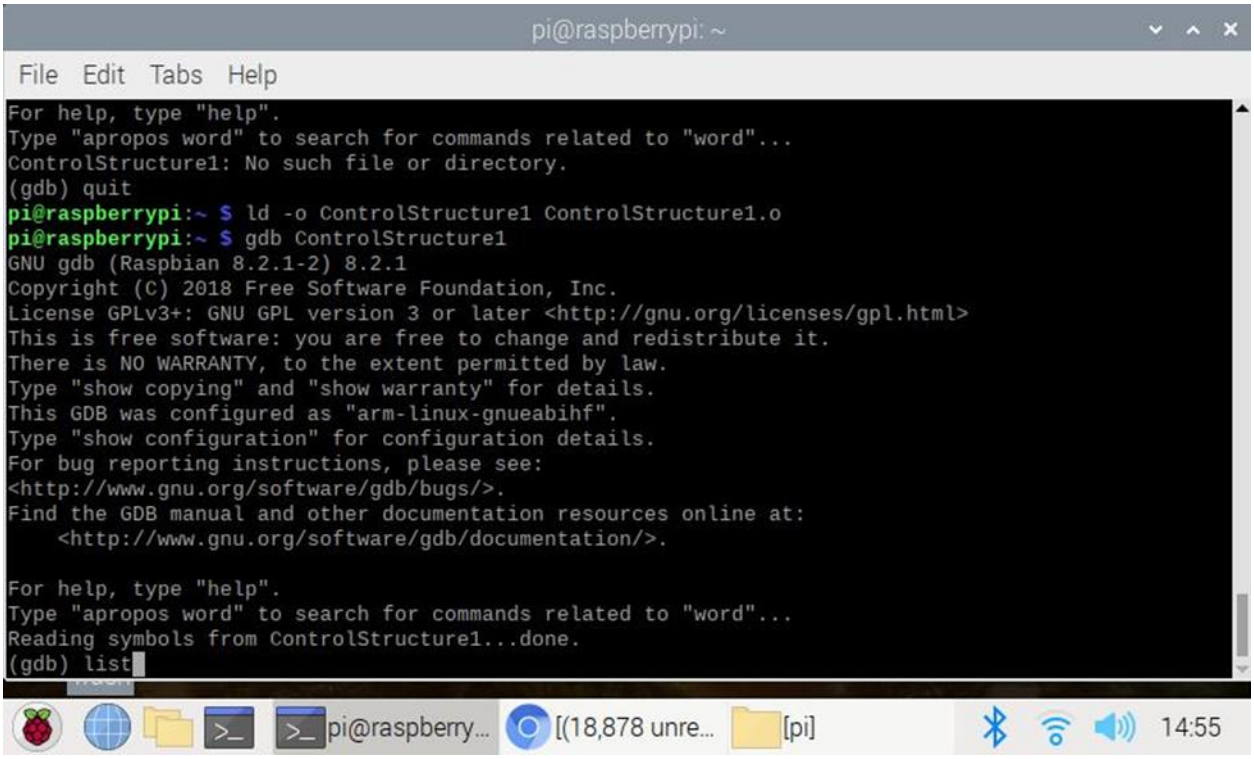
(The end result of execution)

Examined into the registers and flags. My value of r2 is 0 as expected and the z flag value is 10 (the second) which is set as expected.

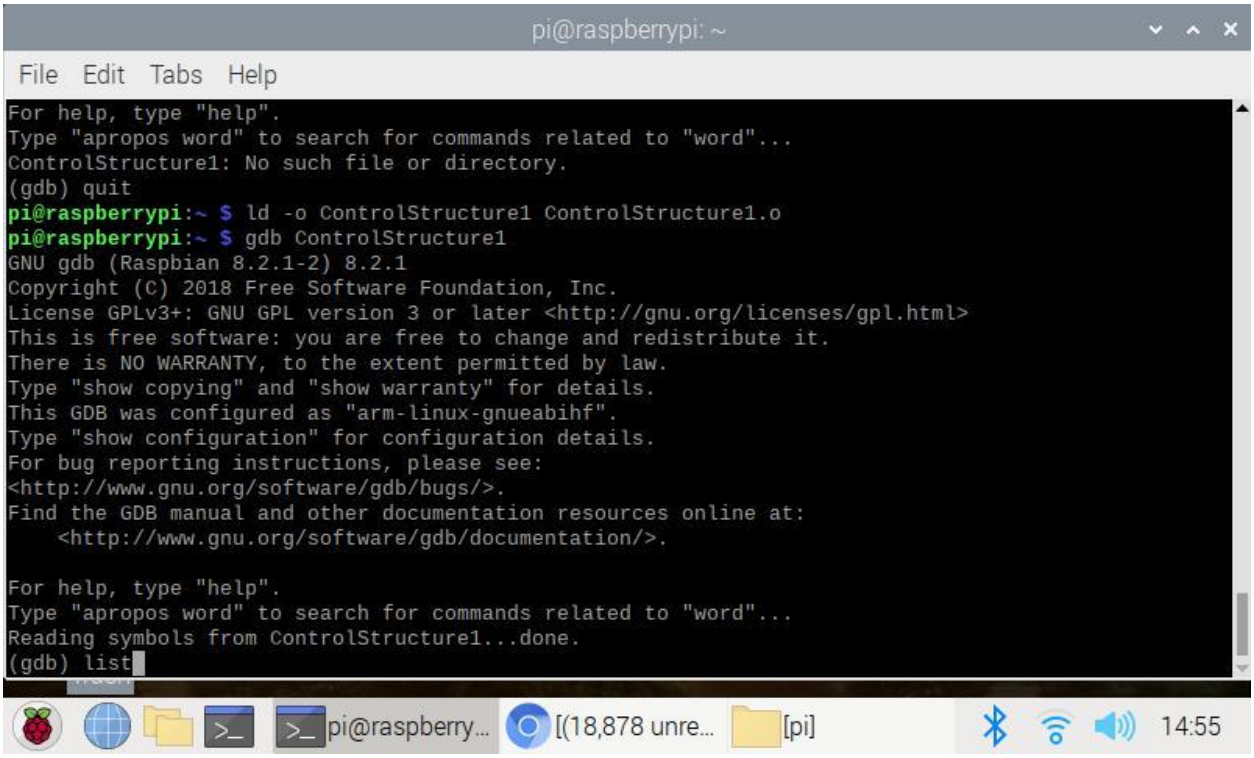
Part3: ControlStructure1.s



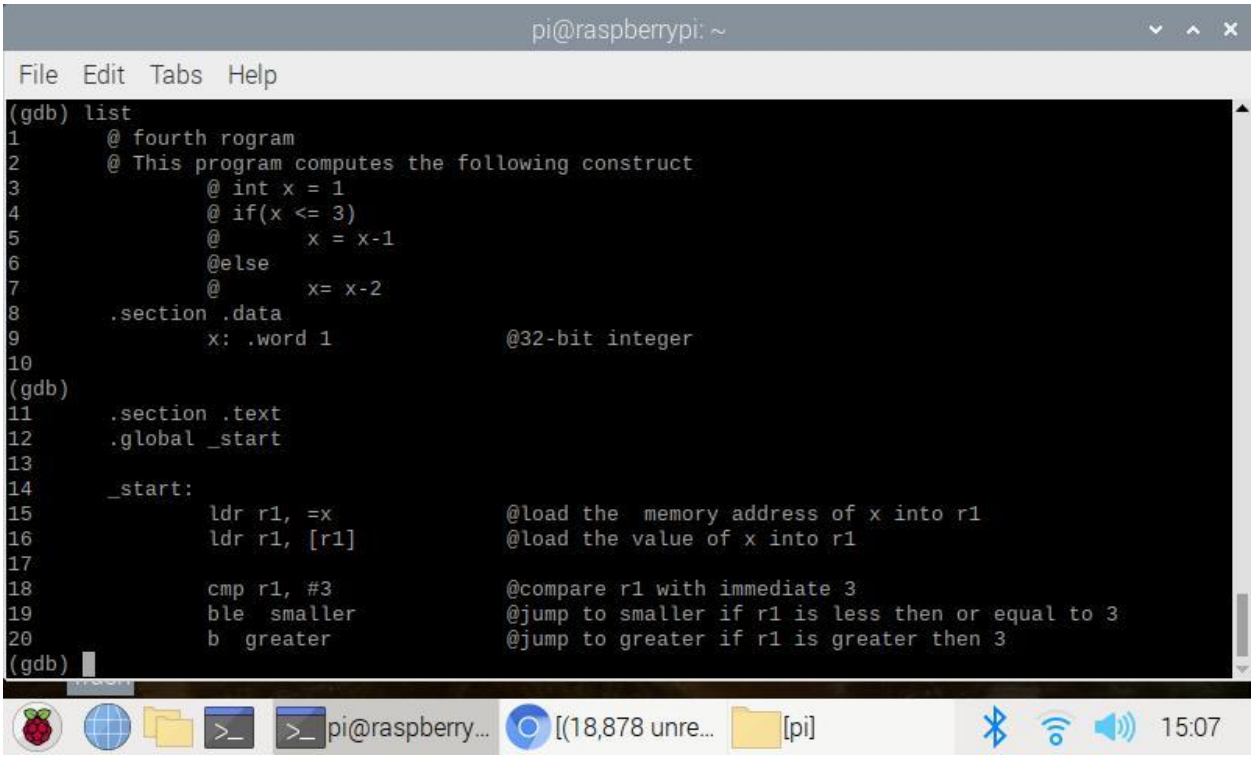
An ARM assembly corresponding to the given if/else condition is programed as above.



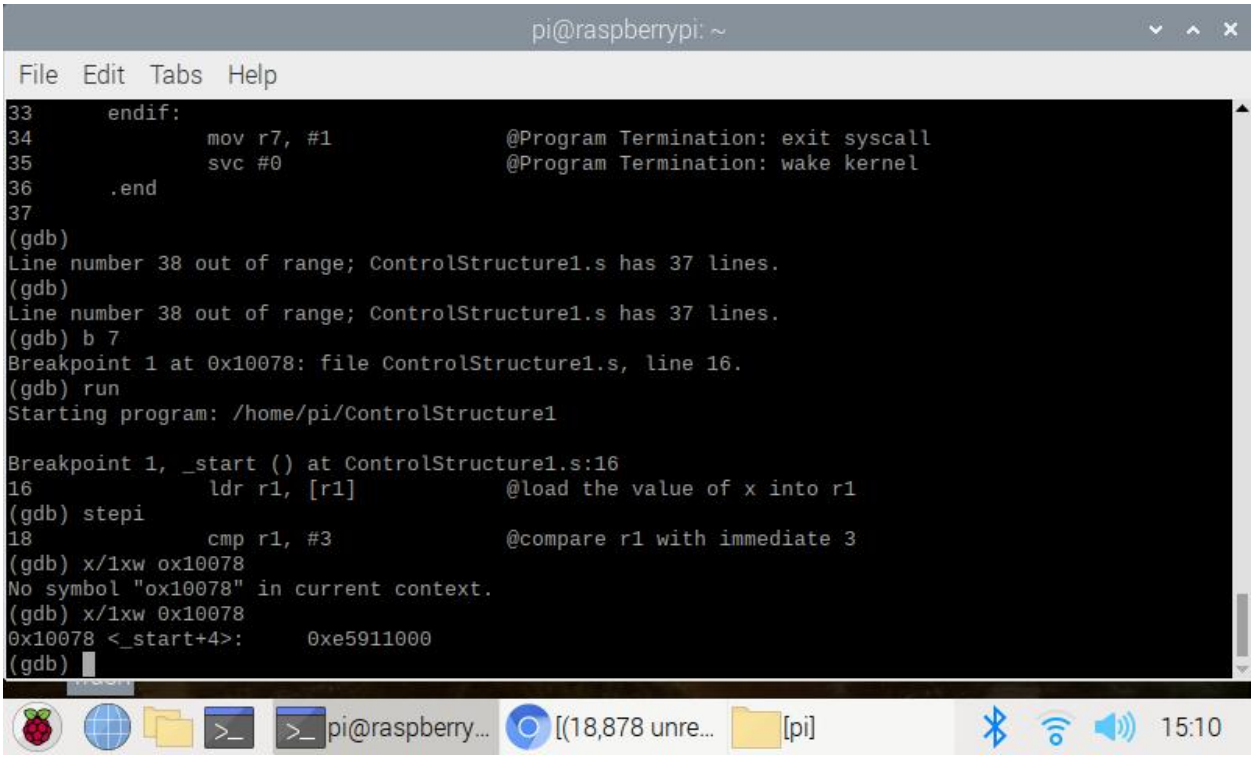
Compiled, linked, ran.



Debugged

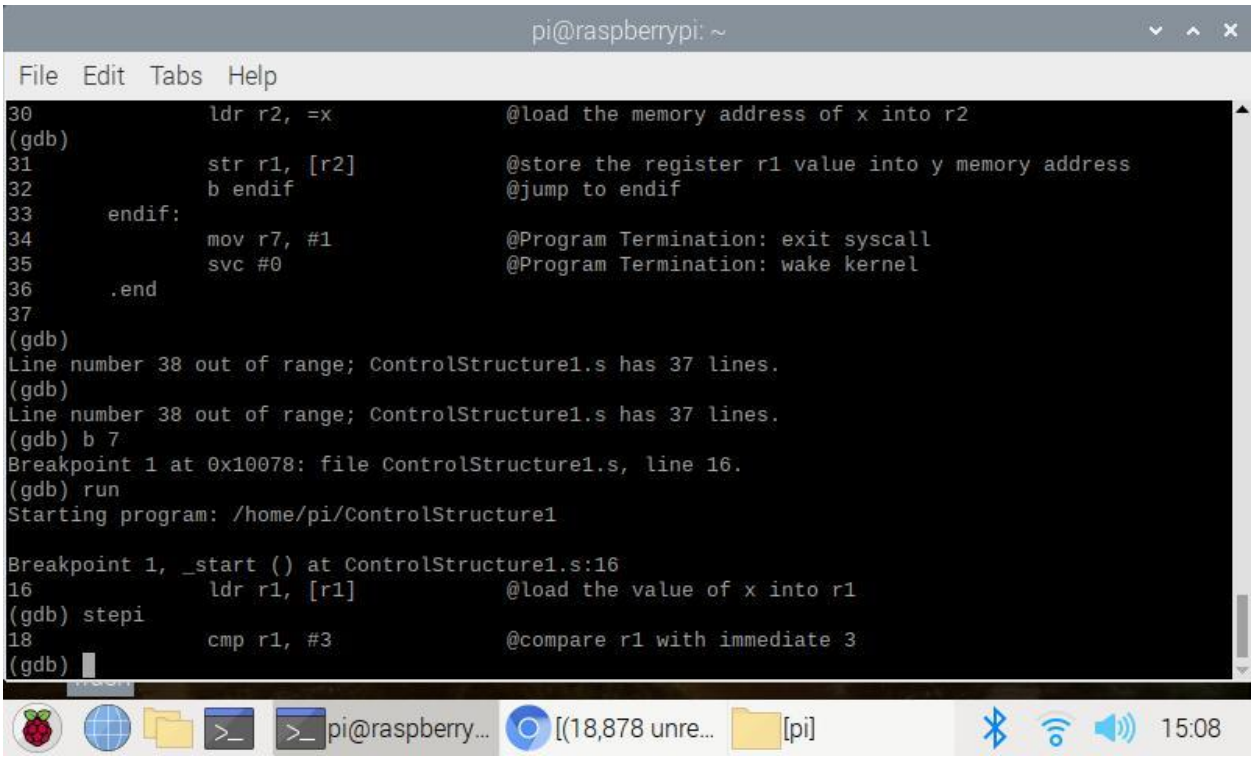


Taken list view to better debugging.

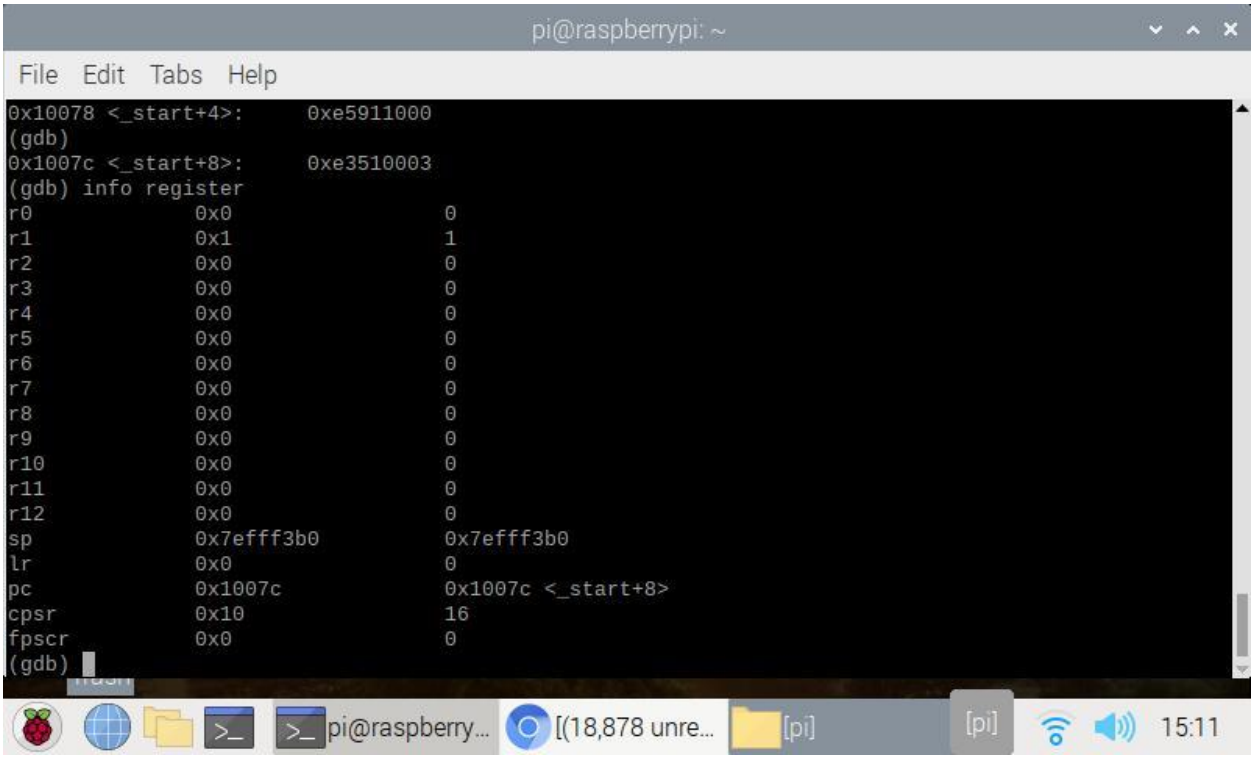


Debugged through breakpoint 7 but taken to line 16

Used step-in to view each lines in the code and examined the value of registers and memory as above.



Viewed into different lines



commanded info register to view into the register.

Our value of x is 0h as expected.

The z flag is cleared as expected because 1-1 = 0

Zoe Kosmicki

Assignment 4

Task 3

- What is race condition?
  The race condition refers to how coded systems behave when dependencies exist in unpredictable elements.
- Why race condition is difficult to reproduce and debug?
  Because they arise due to relative, interdependent interactions, errors that are caused by race hazards can become nonproblems when examined in a more controlled space, like a debugger.
- How can it be fixed? Provide an example from your Project A3
  Looking at spmd2.c, the program we wrote for assignment 2, at first the output wasn't functioning correctly because of how the Raspberry PI's memory is organized. This was fixed by making the variable declarations private, so each thread would be executed in parallel without sharing due to the shared memory. This shows that race conditions are fixed by writing smart and careful code, being sure to trace out your code to be sure there are no hazards.
- Summarize the Parallel Programming Patterns section in the "Introduction to Parallel Computing_3.pdf" (two pages) in your own words (one paragraph, no more than 150 words).

Parallel code can be very difficult to write, and because of this, programmers have created parallel patterns, which are rough guidelines on how to structure parallel code so that it runs and creates the desired output. While different patterns function and are organized in different ways, they all use the same basic building blocks to achieve their end goals.

- In the section "Categorizing Patterns" in the "Introduction to Parallel Computing_3.pdf" compare the following:
    - Collective synchronization (barrier) with Collective communication (reduction)
      Barriers are roadblocks in code that stop processes from continuing until every thread/process has gotten to a specific point. Reduction is an operation where a process can be reorganized to execute in a more efficient way. They're both forms of management and communication within parallel computing.

    - Master-worker with fork join
      Master-worker and fork join are similar, but with master-worker, each "worker" thread or process works in parallel with every other "worker", with the master process finishing after each worker has finished and their individual outcomes have been computed together. In fork join, there are "regions" of parallelism, where a process can start sequentially and branch off to do tasks in parallel, and then rejoin together to continue sequentially.

- Where can we find parallelism in programming?
  In programs (within the code itself), in data (how data is stored and used), and in a machine's resources (how these resources are used and allocated).

- What is dependency and what are its types (provide one example for each)?
  A dependency is when code needs to rely on and within itself to produce an output. The types of dependencies are true dependencies, where a later operation depends on a previous operation (abc = 7; d = abc), anti-dependencies, where a variable that is used in an operation gets changed after the operation (a = 9; b = a - 3; a = 100), and output dependencies, where if the order of the instructions were changed the output would change as well (a = 5; b = a + 7; a = 2).

- When a statement is dependent and when it is independent (Provide two examples)?
  A statement is dependent when it relies on an earlier or later statement for its outcome (b = 7; a = b). A statement is independent when it can execute on its own without needing to rely on other statements (a = 10 + 20; b = 17 + 9).

- When can two statements be executed in parallel?
  When no dependencies exist between the statements and they can be in any order without affecting the outcome.

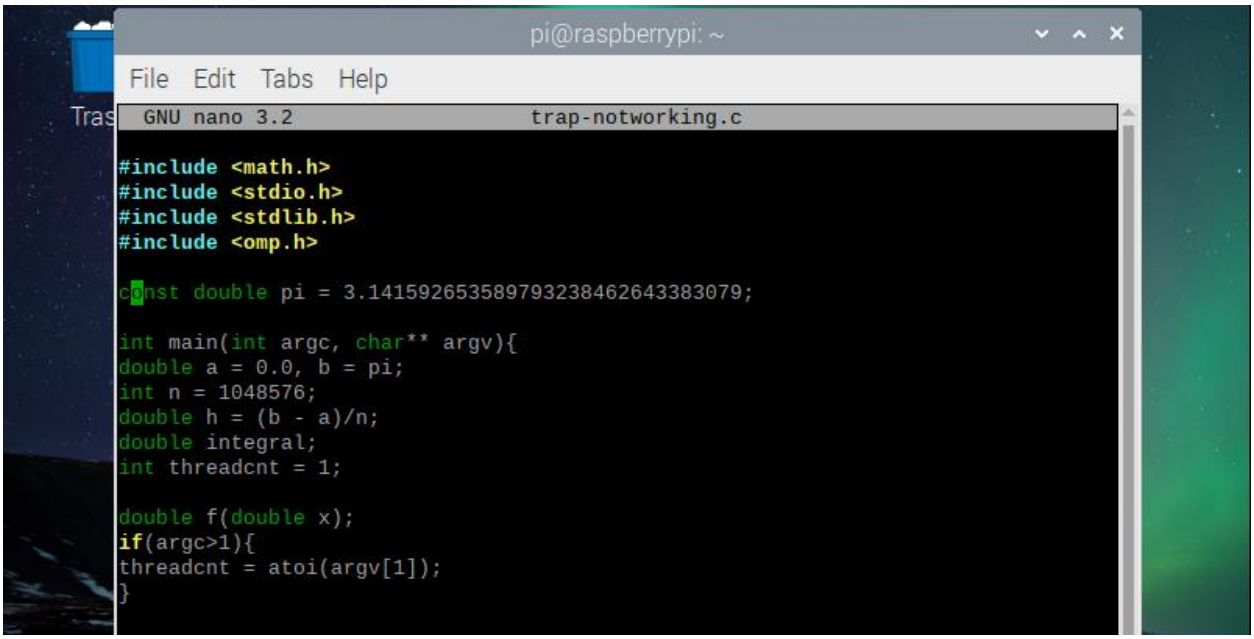
- How can dependency be removed?
  Some dependencies can be removed by rewriting/editing code. However, not all dependencies can be removed.

- How do we compute dependency for the following two loops and what type/s of dependency?
  By examining the IN and OUT sets of the loops and performing dependency analysis to see where dependencies exist in the code. In the first loop, there's true dependency between

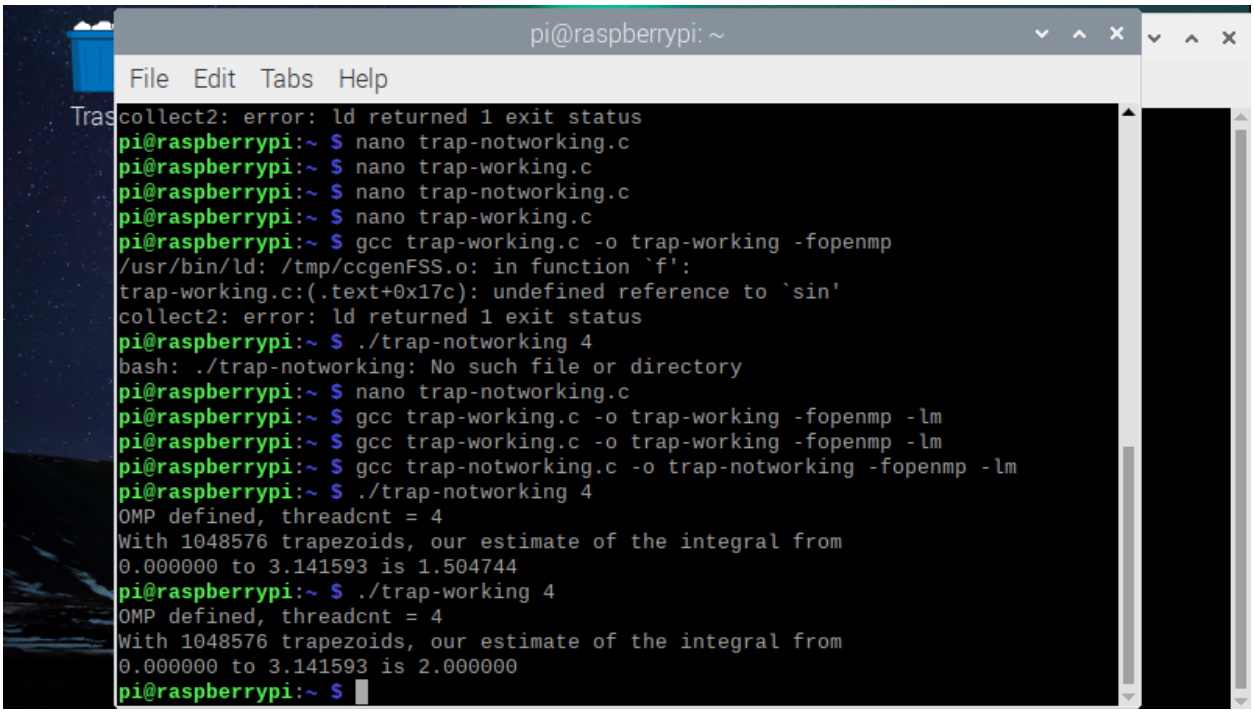
the loop index and a. In the second loop, there's true dependency between the loop index, a, and b.

Task 3 Part B

First I wrote out the code for trap-notworking.c and trap-working.c.

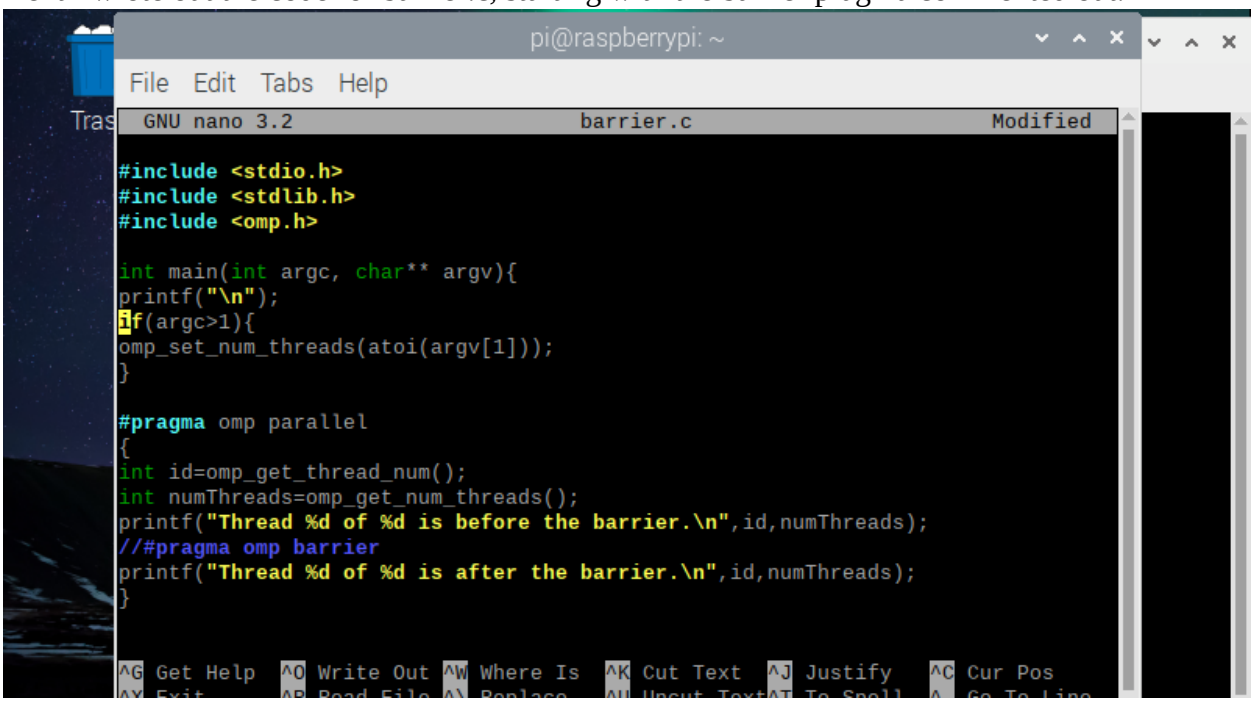


After fixing a few spelling errors and an error having to do with linking the math library, I created executables from both files and ran both with the input 4.



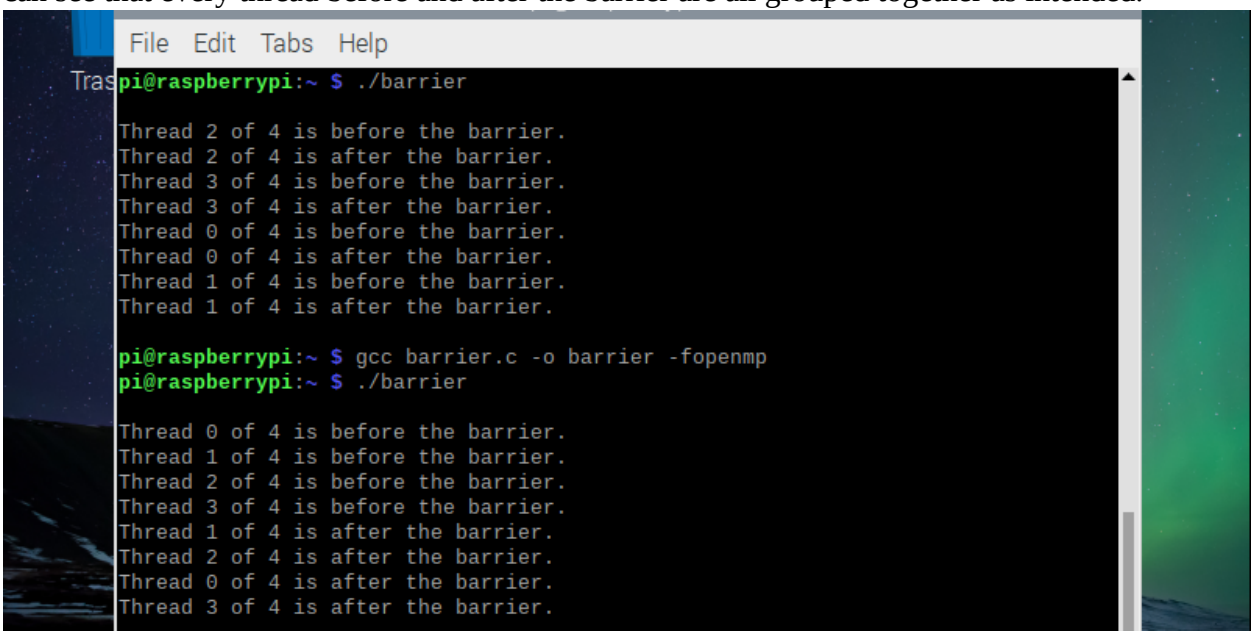
As expected, the notworking file gives an incorrect answer, while the working file gives the correct one.

Next I wrote out the code for barrier.c, starting with the barrier pragma commented out.

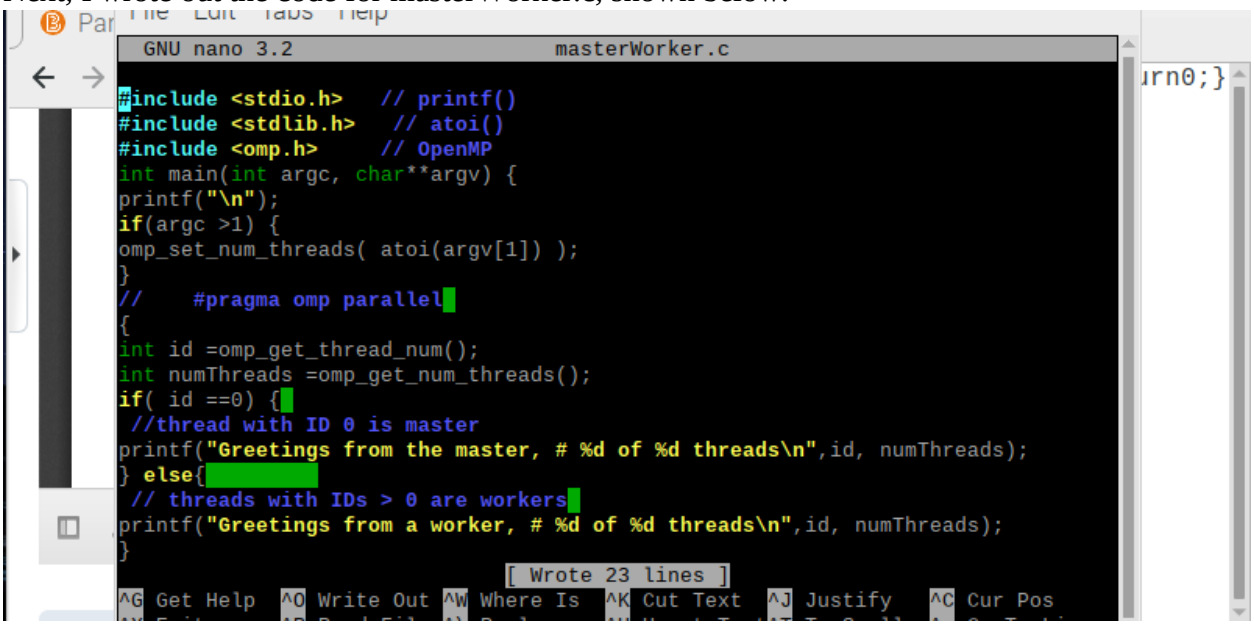


Afterwards, I created the executable file and ran it as-is, and then with the barrier pragma uncommented, following the same steps. Without the barrier, the threads have no set order to execute in, so we see the befores and afters being mixed up. With the barrier uncommented, we

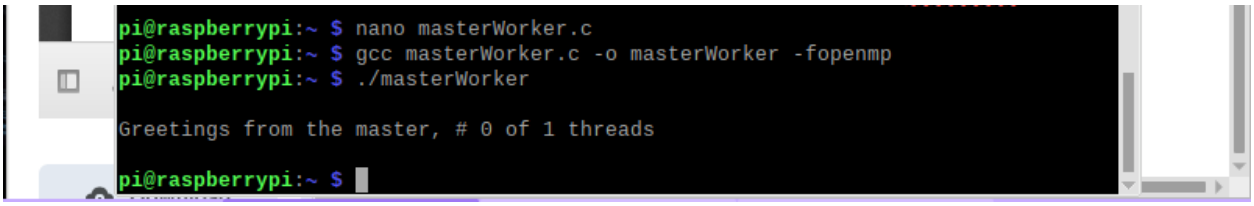
can see that every thread before and after the barrier are all grouped together as intended.



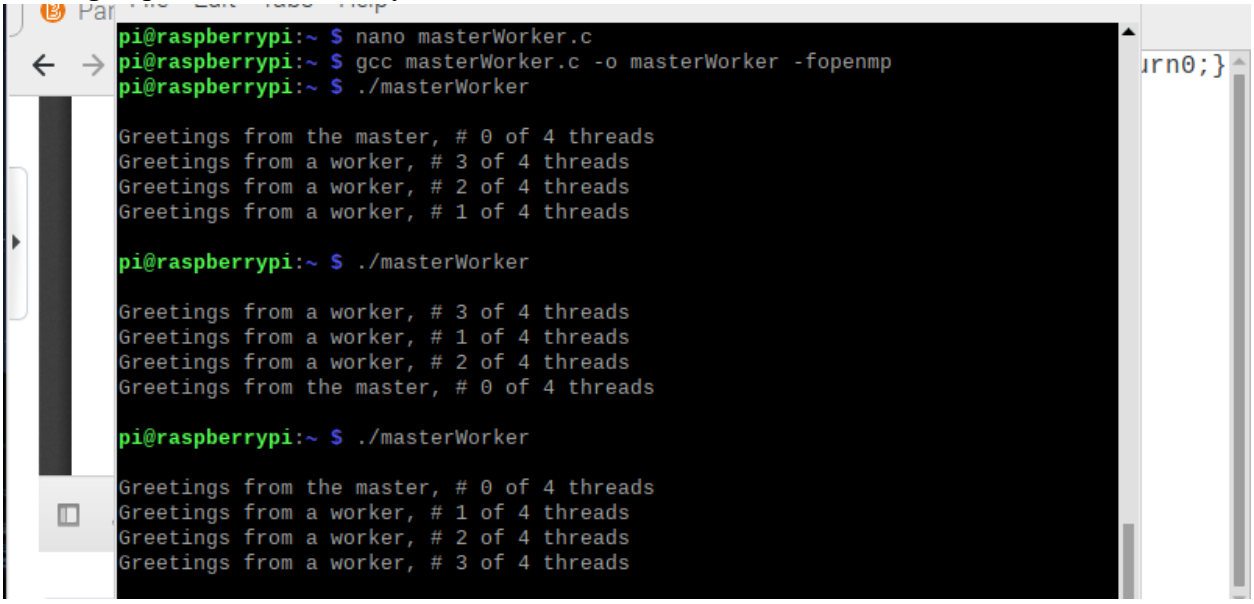
Next, I wrote out the code for masterWorker.c, shown below.



With the parallel pragma commented out, I linked and ran the executable, and got the output shown below.



This makes sense, since master-worker is a type of parallel pattern, so if the program isn't running in parallel, there's only the master thread and no workers.



After uncommenting the parallel pragma and running the program again a few different times,

we can see the master thread appears once, and each worker thread having their unique threads executed successfully.

Hyoungjun Lee

Task 3
Foundation
What is race condition?
-this condition is when a software, electronics or other system where the system's function is in dependent on sequence, if you do not use properly, it causes a bug
What race condition is difficult to reproduce and debug?
-when debugging race conditions, we cannot correct the programming and hard to reproduce time and energy in software development, interdependent interactions
8p) How can it be fixed? Provide an example from your Project_A3(see spmd2.c)

-when we looking at spmd2.c the output was not coming out correctly, because how Raspberry PI organized memory, to fix this issue, we commented out line 5 and declared id and numThreads as the int data type

Change to below

```
#pragma omp parallel {
int id = omp_get_thread_num();
int numThreads = omp_get_num_threads();
printf("Hello from thread %d of %d\n", id, numThreads);
}
```

Summaries the Parallel Programming Patterns section in the "Introduction to Parallel Computing_3.pdf"(two pages) in your own words (one paragraph, no more than 1 50 words).-(12p)

-parallel programming divide into two categories , implementation and Algorithmic. Parallel code can be difficult to write, so programmers need to created parallel patterns, roughly guideline, This pattern uses both openMP and Message Passing interface. Concurrent execution mechanism divided into two major categories, Process and Thread control patterns and coordination patterns.
In the section "Categorizing Patterns" in the "Introduction to Parallel Computing_3.pdf" comparethe following:
oCollective synchronization (barrier) with Collective communication (reduction)

-collective communication, all the process have a specific point before execution, reduction is an operation that process can be organized to execute in efficient way, uses concurrent execution mechanism for parallel execution, in collective Synchronization, blocks all the processes until processes are reached specific point, it uses MPI_Barrier()function, this is type of parallel application of computing

oMaster-worker with fork join

-Master-work, main process is being divided into small chunks, distributed to several worker processes, in Fork join , used to execute parallel light weight processes and threads
-(26p) Dependency: Using your own words and explanation, answer the following:
(3p) Where can we find parallelism in programming?

-in programs, server machines, virtual reality, data bases

(6p) What is dependency and what are its types (provide one example for each)?

-input of one execution depends on the output, other elements, we have true dependence, output dependence, anti dependence

(3p) When a statement is dependent and when it is independent (Provide two examples)?

-statement is dependent when its values is getting computed, or it relies on statement for some output

Example: a = 5 b = 10 c = a+b c = a + b is dependent on a = 5 b = 10 statements

statement is independent when we do not need previous values

a = 5 b = 10 c = 30 + 20 where c = 30+10 does not need value of a and b

(3p) When can two statements be executed in parallel?

-two statement in parallel can be executed if they don't share any common data item, no dependency between cycle

(3p) How can dependency be removed?

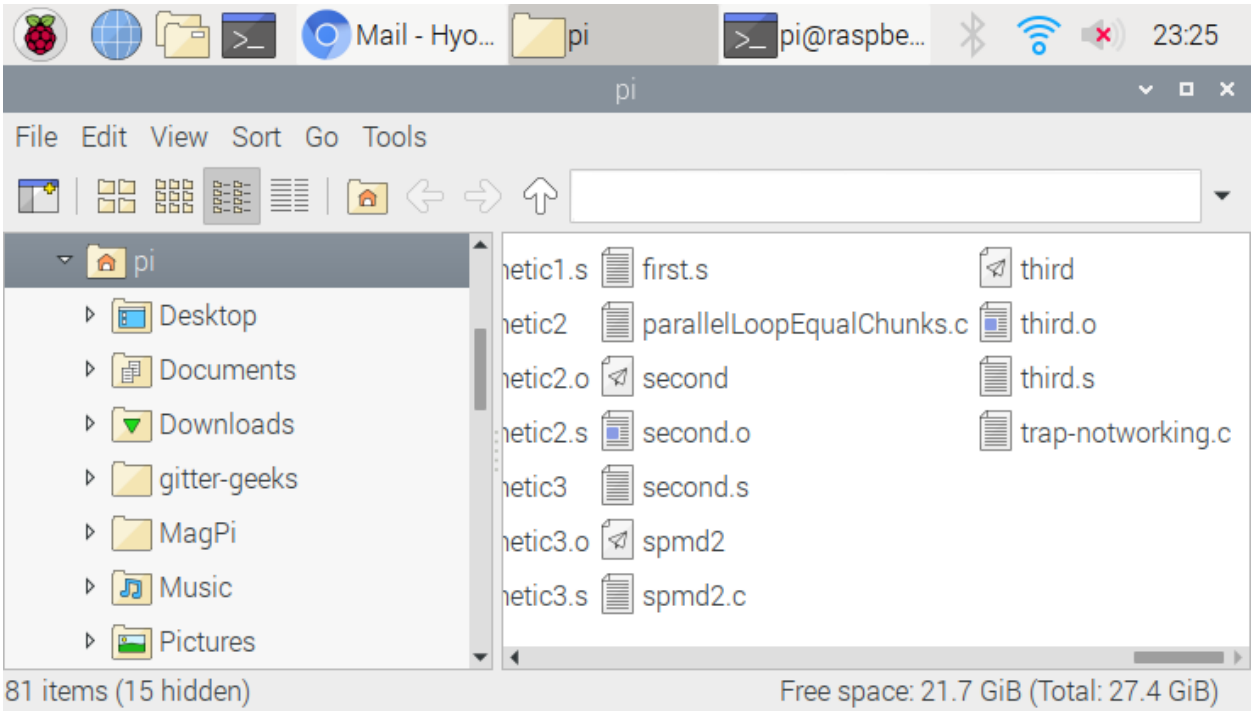
-rearranging and eliminating statements before execution

(8p) How do we compute dependency for the following two loops and what type/s of dependency

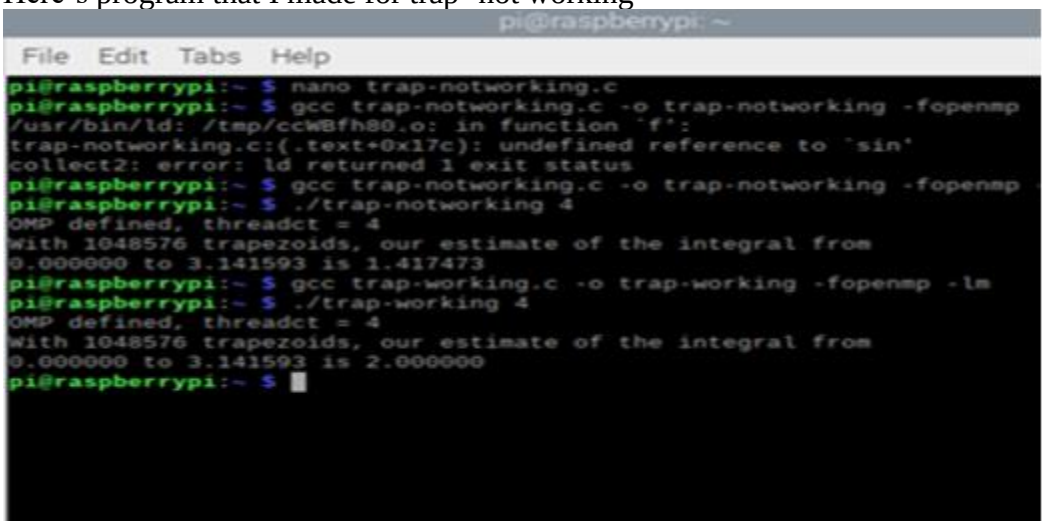
-examining in and out the loops, use two basic strategies to find dependency for the statement

Study the relationship between statement, unrolling of the loop into iterations, In the first loop, there is true dependency between loop index and a, and loop index a and be in second loop

Part B

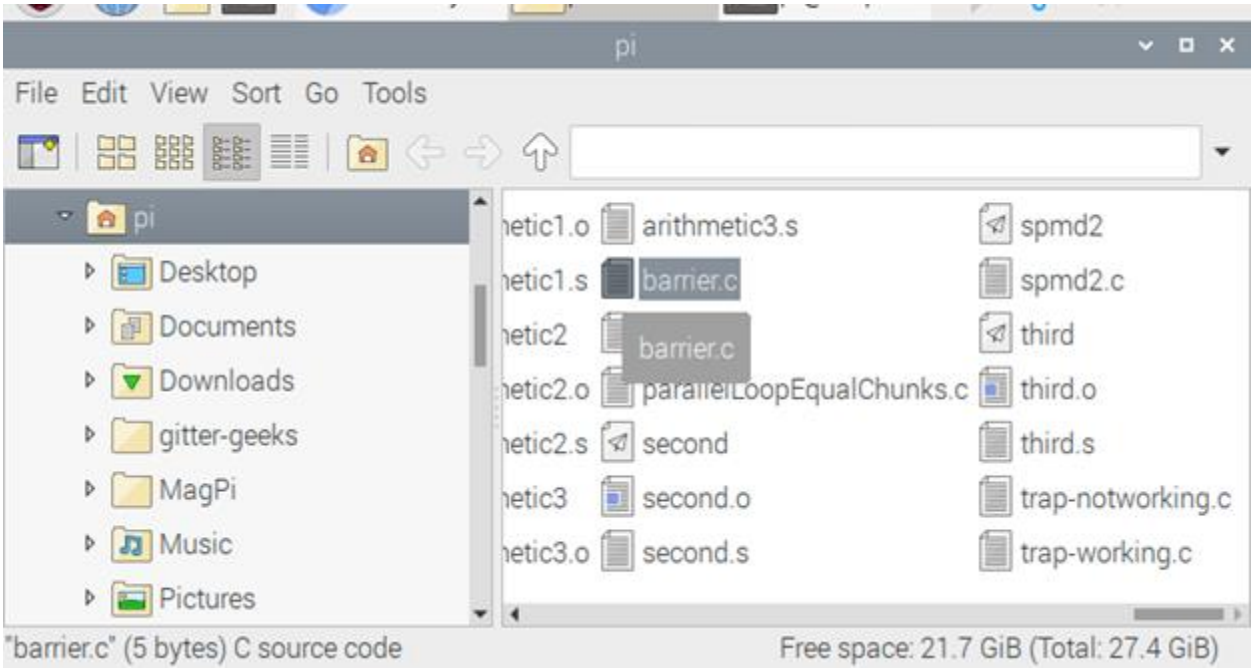


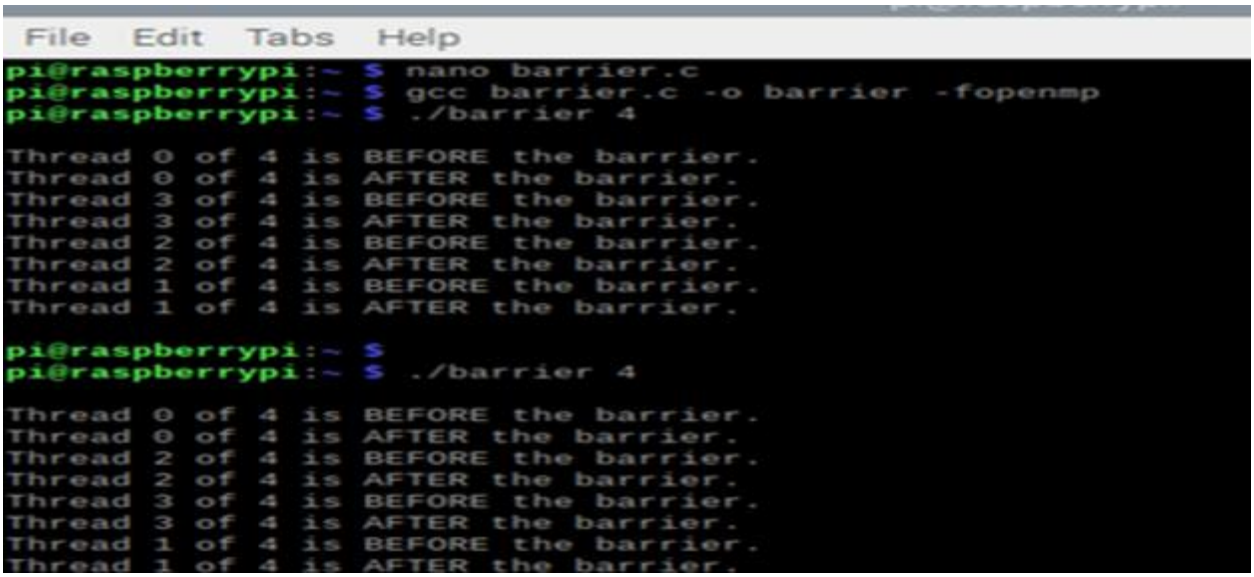
Here's program that I made for trap- not working



Trap-notworking compiled successfully and ran
But the result was not correct since value of our integral at the first
After fixing the problem about integral, I ran it again with gcc trap-working-c
We get the correct result output

Reason for giving an incorrect output is because of race condition. So we used \ at the end of #pragma, save the last iteration.





Without barrier above,
There is a difference between without barrier and with barrier, thread that without barrier print out before and after statement 1 by 1, but when running with barrier , all the threads stopped, until all other threads caught up

```
pi@raspberrypi: ~
File  Edit  Tabs  Help
pi@raspberrypi:~ $ ./barrier

Thread  0 of 4 is BEFORE the barrier.
Thread 0 of 4 is AFTER the barrier.
Thread  1 of 4 is BEFORE the barrier.
Thread 1 of 4 is AFTER the barrier.
Thread  2 of 4 is BEFORE the barrier.
Thread  3 of 4 is BEFORE the barrier.
Thread 3 of 4 is AFTER the barrier.
Thread 2 of 4 is AFTER the barrier.

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

Thread  1 of 4 is BEFORE the barrier.
Thread  3 of 4 is BEFORE the barrier.
Thread  2 of 4 is BEFORE the barrier.
Thread  0 of 4 is BEFORE the barrier.
Thread 1 of 4 is AFTER the barrier.
Thread 2 of 4 is AFTER the barrier.
Thread 3 of 4 is AFTER the barrier.
Thread 0 of 4 is AFTER the barrier.
```

This is with the barrier

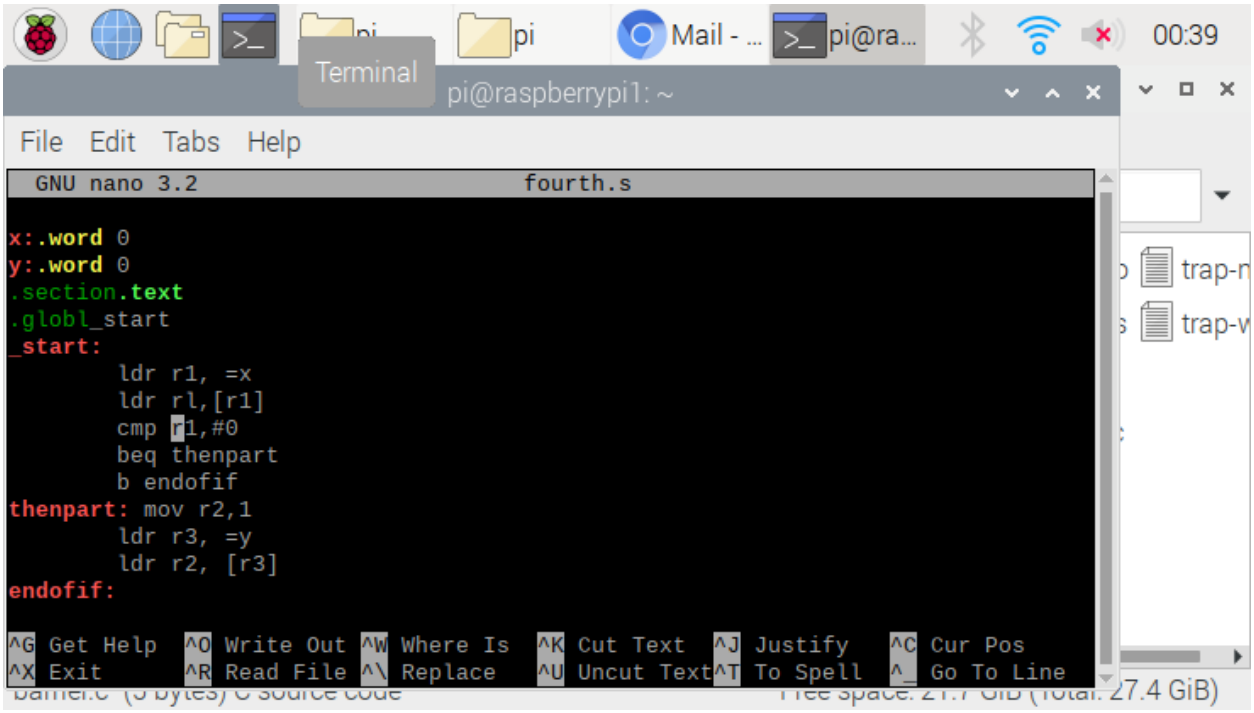
```
pi@raspberrypi:~ $ gcc masterWorker.c -o masterWorker -fopenmp
pi@raspberrypi:~ $ ./masterWorker

Greetings from  a worker, # 1 of 4 threads
Greetings from  a worker, # 2 of 4 threads
Greetings from  a worker, # 3 of 4 threads
Greetings from the master, # 0  of 4 threads

pi@raspberrypi:~ $ ./masterWorker 4

Greetings from  a worker, # 3 of 4 threads
Greetings from the master, # 0  of 4 threads
Greetings from  a worker, # 2 of 4 threads
Greetings from  a worker, # 1 of 4 threads
```

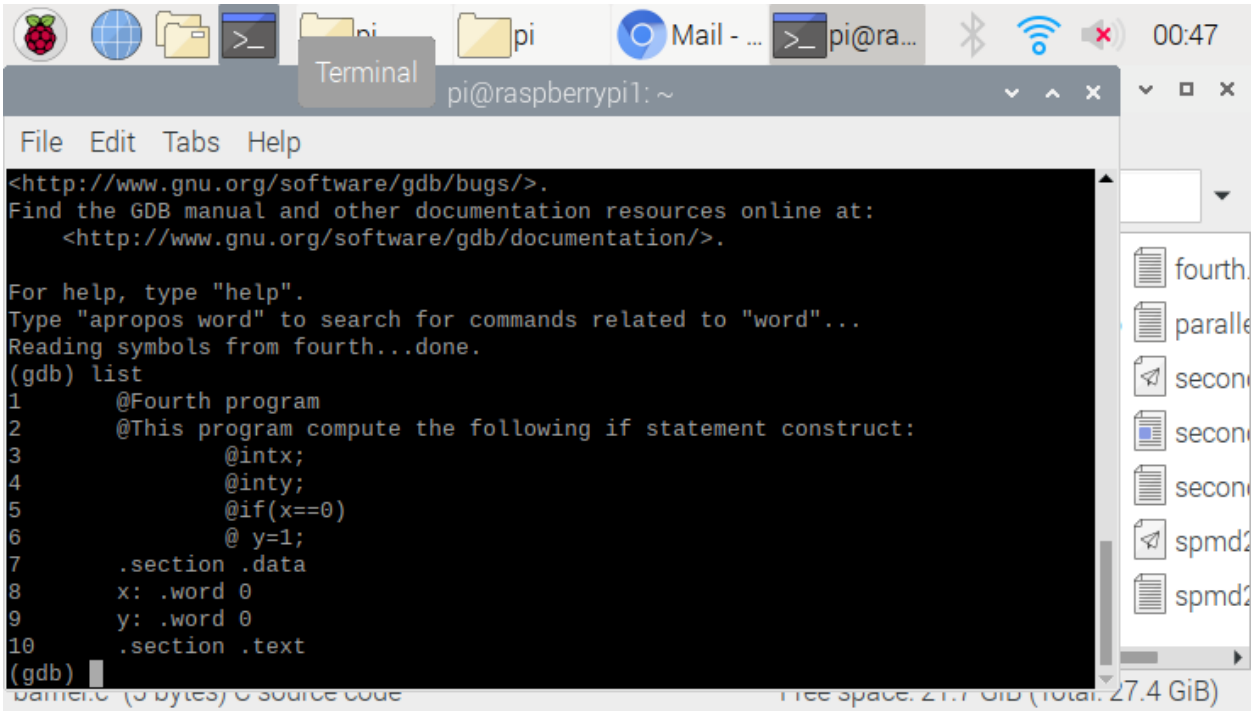
Worker master without pragma and with pragma

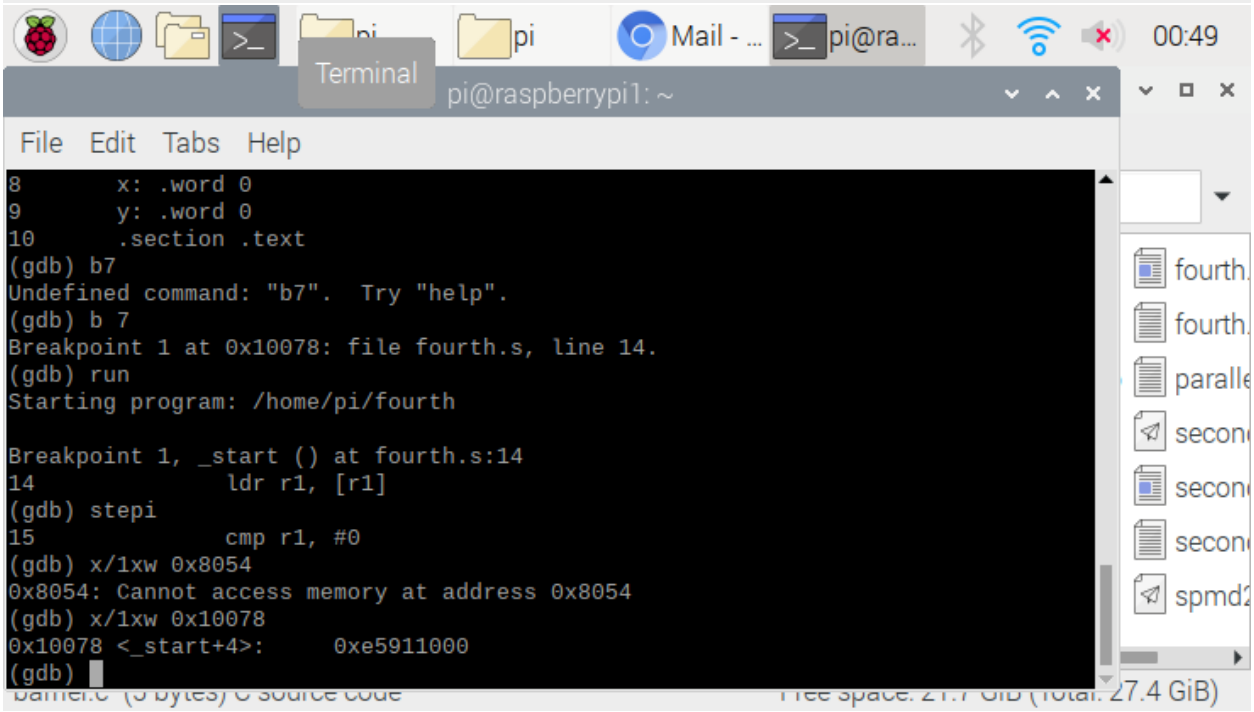
After creating code executable, ran the code, one with pragmas, there's no thread are being used, just sequential computation. Without parallel, I can see only 1 line output of the master

Task4

First, I wrote out fourth.s, and to do debug, I assemble it and linking the program



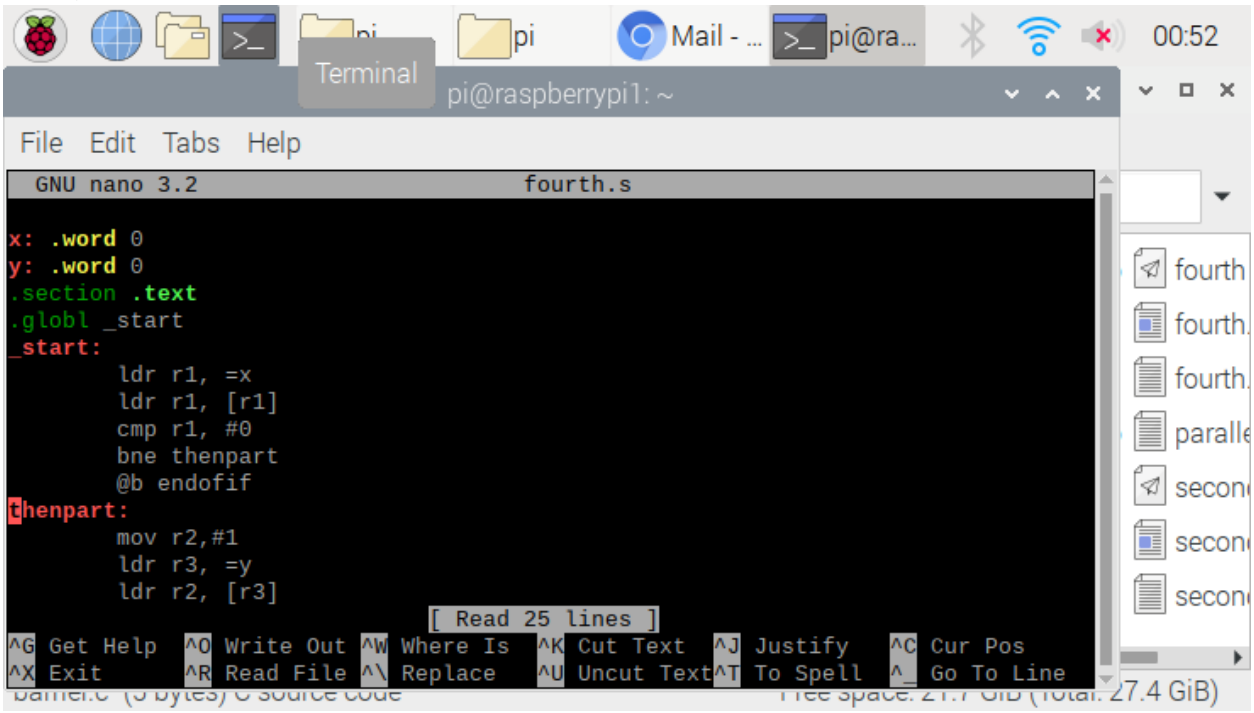




And I set up the break point at line 7, where the instruction started, and it will go through the whole program until .end

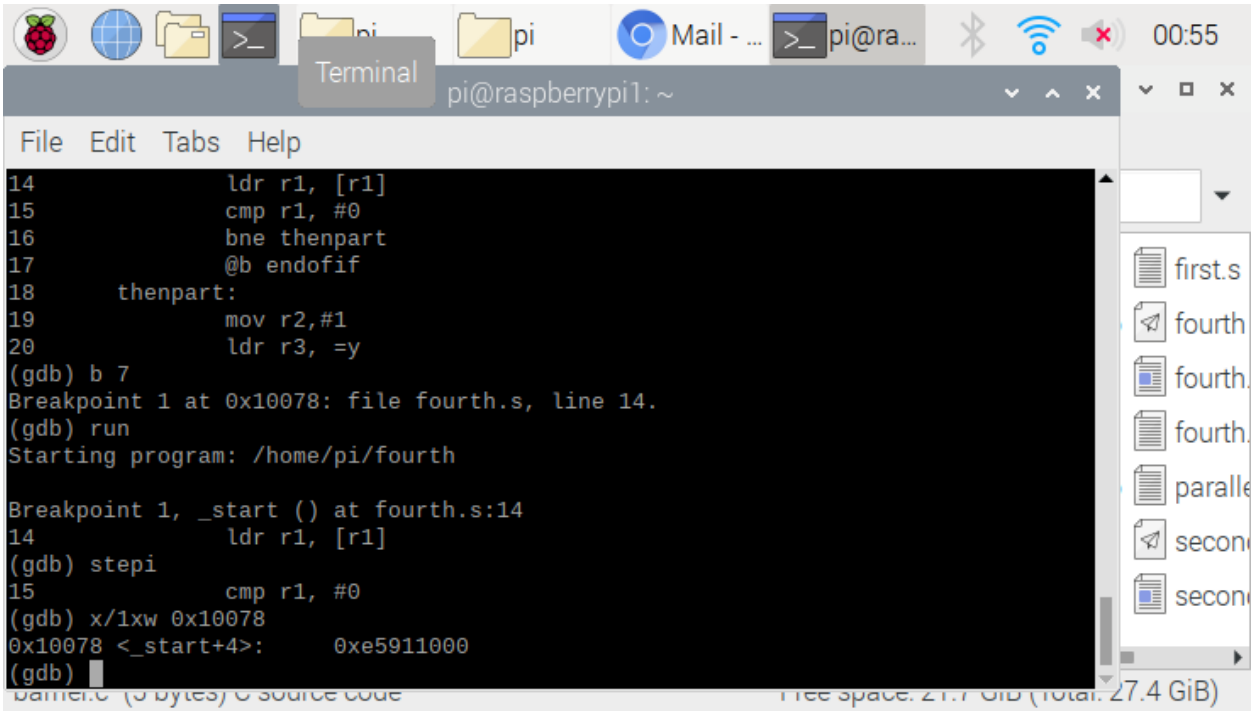
And using stepi, and x/1xw 0x10078 at the point, and hex value will be 0xe5911000

Part 2)



improved code for fourth.s

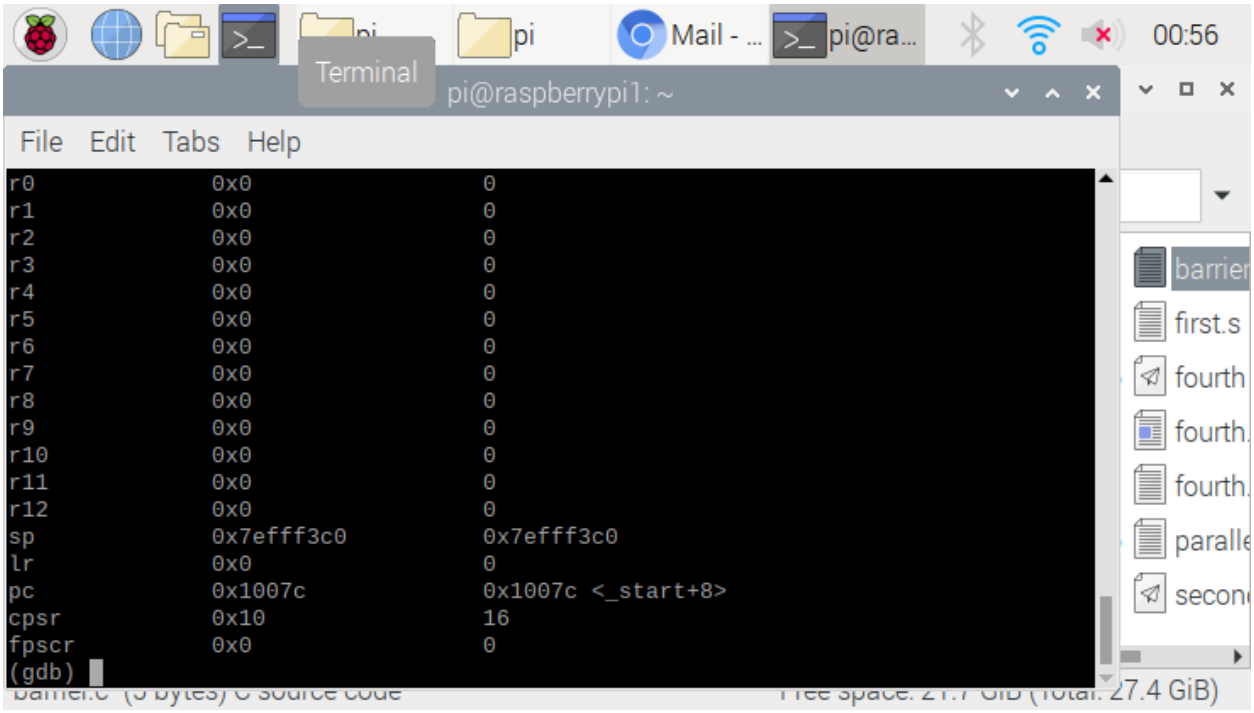
We corrected code for the better look and efficiently to avoid back to back, we removed b and change beq to bne, using Demorgan's law



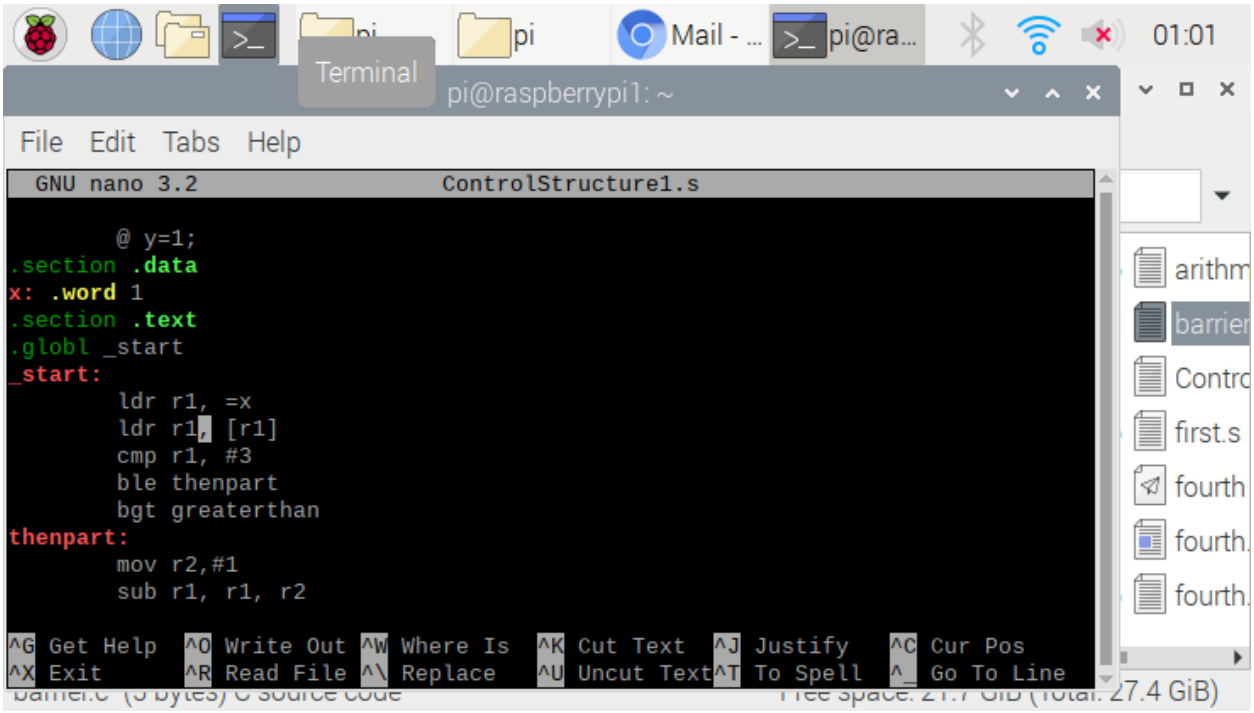
We compiled and link and debug

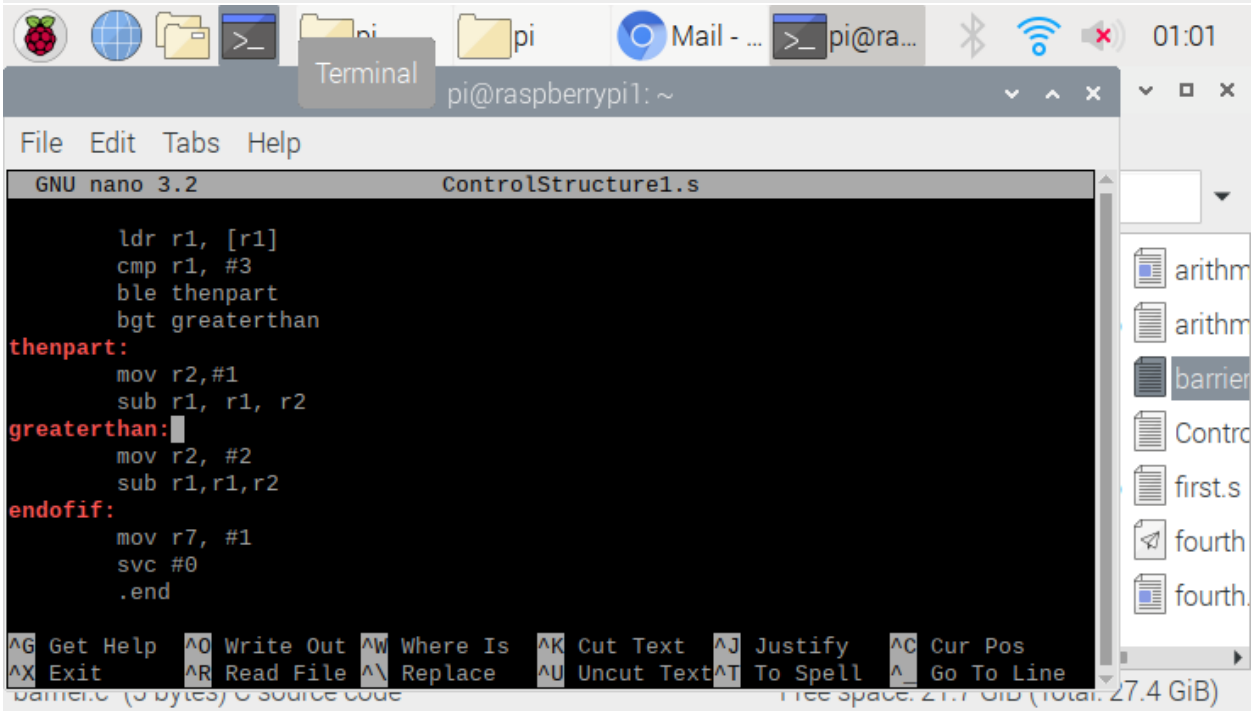
And we examining the flag using info register

My value of r2 is 0 as expected and zflag is 10, which is set

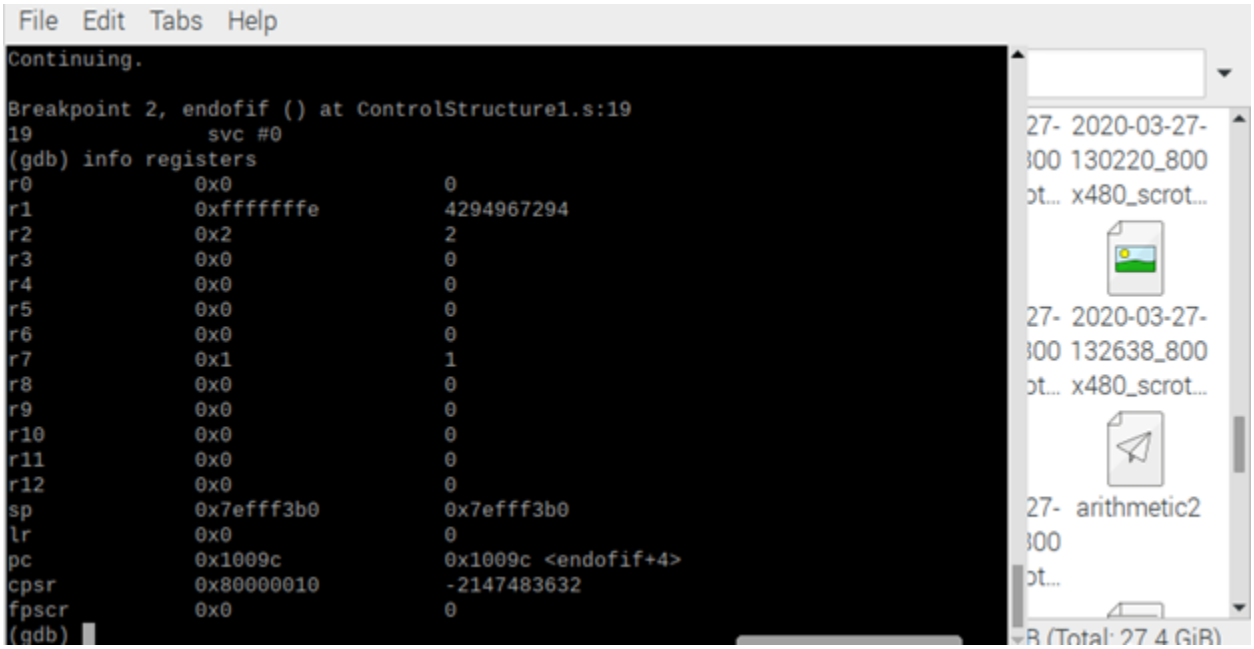


Part3


Terminal
pi@raspberrypi1: ~
File Edit Tabs Help
GNU nano 3.2 ControlStructure1.s
@ y=1;
.section .data
x: .word 1
.section .text
.globl _start
_start:
ldr r1, =x
ldr r1, [r1]
cmp r1, #3
ble thenpart
bgt greaterthan
thenpart:
mov r2,#1
sub r1, r1, r2
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Spell ^_ Go To Line


Terminal
pi@raspberrypi1: ~
File Edit Tabs Help
GNU nano 3.2 ControlStructure1.s
ldr r1, [r1]
cmp r1, #3
ble thenpart
bgt greaterthan
thenpart:
mov r2,#1
sub r1, r1, r2
greaterthan:
mov r2, #2
sub r1,r1,r2
endofif:
mov r7, #1
svc #0
.end
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Spell ^_ Go To Line

I wrote code above, breakpoint 7 and 19, and I examine the register after end of program



The value of zero flag is 0 in this example

VEDANSI PARSANA

TASK 3A)

- Race condition:

❖ What is race condition?

→ A race condition (race hazard) it's the condition of an electronics, software, or other system where the system's functional behavior is dependent on the sequence or timing of other uncontrollable events. When programmer wants to do any event and it doesn't happen than it becomes a bug.

❖ Why race condition is difficult to reproduce and debug?

→ Race condition is difficult to reproduce and debug because debugging race conditions in C/C++ Bugs that are hard to reproduce pull up time and energy of any software development team. Therefore, one cause of these bugs can be race conditions, which causes unpredictable behavior and make getting a reliable bug report which is nearly impossible. The difficulty in locating the race conditions is because nothing really goes wrong with the program unless a trigger is activated to do so.

❖ How can it be fixed? Provide an example from your Project_A3 (see spmd2.c)

→ In order to avoid the race condition from happening is to plan and write the code design for the program.

Here is an example from the spmd2.c: one of the thread's id is appearing more than once.

When compiled and ran, the program printed out something similar to this:

Hello from thread 1 of 4

Hello from thread 2 of 4

Hello from thread 1 of 4

Hello from thread 2 of 4

To fix this issue, we commented out the initial declaration in line 5 and declared id and numThreads as the int data type.

```
#include <stdio.h>
#include <omp.h>
#include <stdlib.h>
int main(int argc,char** argv) {
//int id, numThreads;
printf("\n");
if (argc > 1) {
omp_set_num_threads( atoi(argv[1]));
}
#pragma omp parallel
{
int id = omp_get_thread_num();
int numThreads = omp_get_num_threads();
printf("Hello from thread %d of %d\n",id,numThreads);
}
printf("\n");
return 0;
```

- Summaries the Parallel Programming Patterns section in the "Introduction to Parallel Computing_3.pdf" (two pages) in your own words

→ There are to main topics discussed in the parallel programming patters that is strategies and concurrent execution mechanism. Strategies apparatuses parallel programming and further divides into two categories Implementation and Algorithmic. Implementation determines how are the tasks of program are being processed. Algorithmic determines how and when tasks should be executed. Concurrent Execution Mechanism, which relates to the hardware where different operations executes concurrently. The Concurrent Execution Mechanism are divided into two major subcategories: Process/Thread control patterns and Coordination patterns. The Process/Thread control patterns controls which thread to process and when will this happen at the runtime. Third implementation involves hybrid computation with combination of Strategies and Concurrent Execution Mechanisms. This pattern uses both OpenMP and Message Passing Interface.

- In the section "Categorizing Patterns" in the "Introduction to Parallel Computing_3.pdf" compare the following: Collective synchronization (barrier) with Collective communication (reduction) and Master-worker with fork join.

→ In **Collective communication**, all the processes are reach to a specific point before executing. It acts as a reduction as once process of the communicator collects data from all other processes and performs an operation to find the result. It uses MPI_Reduce () function. In **Collective Synchronization**, it blocks all the processes until a specific synchronization point is being reached. It acts as a barrier since it blocks the process until all other processes are being reached to synchronization point successfully. It uses MPI_Barrier () function. In **master-worker** pattern, a main process is being divided into small pieces which in turn being distributed to several worker processes whereas in **Fork-Join** pattern which is used to execute parallel light weight processes and threads.

- Dependency: Using your own words and explanation, answer the following:

❖ Where can we find parallelism in programming?

→ Parallelism is a mechanism that enables programs to run faster by performing several computations at the same time by incorporating hardware with multiple CPU's. mobiles, databases, servers, virtual reality and man more.

❖ What is dependency and what are its types (provide one example for each)?

→ Dependency itself tells that one element is depending on other elements. basically, all our programs will have this dependency in-built. in case of DBMS, we will have dependencies like trivial, non-trivial, multivalues, transitive etc.

❖ When a statement is dependent and when it is independent (Provide two examples)?

→ A statement is dependent if its values is getting computed using previously assigned or computed values.

example: a=10 b=20 c = a + b here statement c = a + b is dependent on a=10 and b=20 statements

A statement is not dependent if its values are independent of previous values.

example: a=10 b=20 c=45+65-90 printf ("hello") here print statement and c=45+65-90 statements are independent of other statements i.e. a=10 and b=20

❖ When can two statements be executed in parallel?

→ two statements can execute in parallel if they don't share any common data item that is getting updated in two statements.

❖ How can dependency be removed?

→ By making multiple statements i.e. for example by applying Normalizations like 1nf, 2nf, 3nf, bcnf, 4nf, 5nf in dbms which creates multiple tables with less data.

❖ How do we compute dependency for the following two loops and what type/s of dependency?

→Here in the leftmost for loop, in instruction S1: a[i] = i, the dependency exist is

a[i] is dependent on variable i and since a[i] value will be written only after reading variable i, hence a[i] is dependent on variable i and the dependency type is RAW which is also called Real dependency. Hence the dependency is a[i] dependent on i with RAW I.e. Real dependency.

In the rightmost for loop, in instruction S1: a[i]= i, a[i] is dependent on variable i with RAW I.e. Real dependency and in instruction S2: b[i] = 2*i, b[i] is dependent on i with RAW.

RAW = Read After Write.

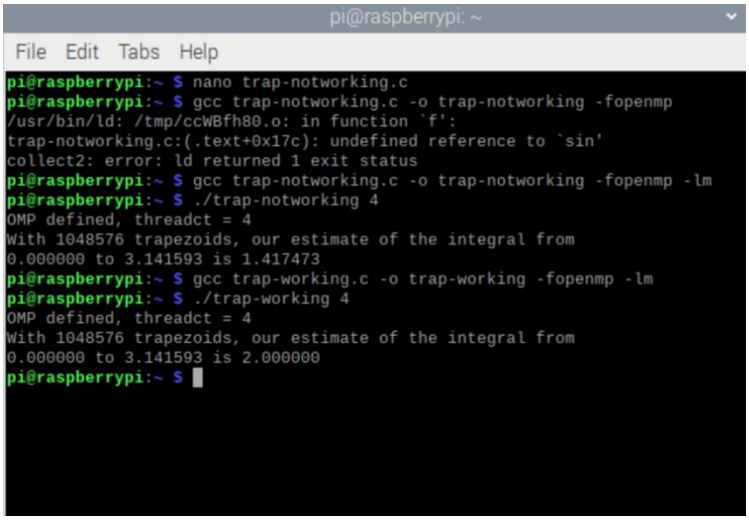
Hence dependencies are:

1. a[i] dependent on i with RAW.

2. b[i] dependent on i with RAW.

Task 3B)

The first part of parallel programming task four, I created both files and executables for trap-notworking and trap-working. Then after their creation I created the executables for each one, however due to the use of the sin method, I had to include -lm at the end of the gcc creation command.

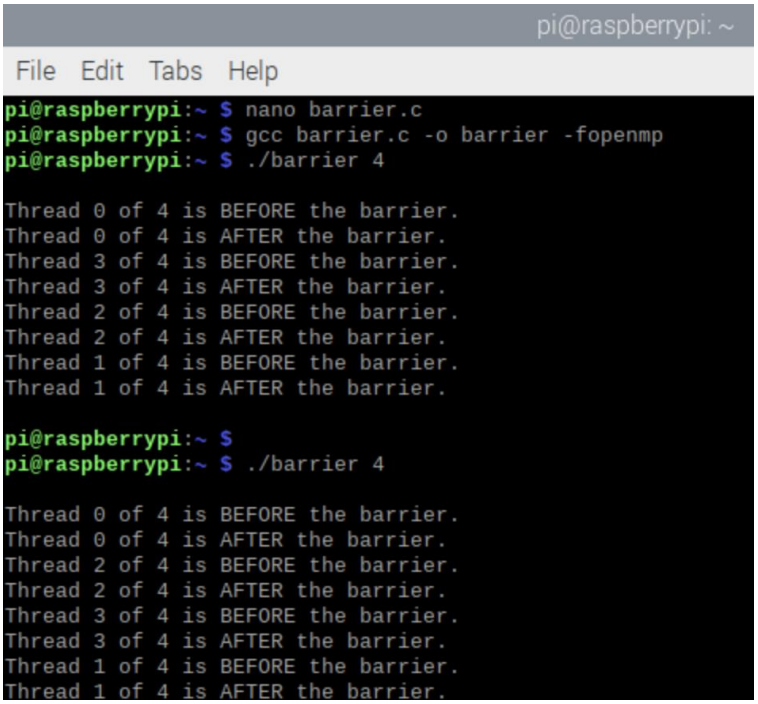


From the output of the programs I can see that trap-notworking fails to receive the correct number, the difference between #pragma omp parallel for with \ and without, is that using it will save the last iterations data for use outside the #pragmas… clause.

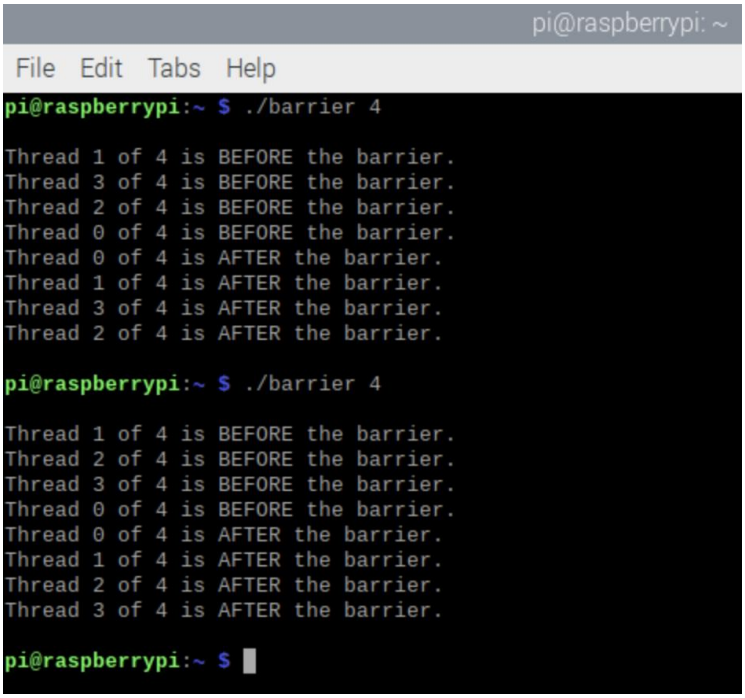
Barrier:

The barrier class had to tests to run, using it without the included #pragma omp barrier and using it with it.

(without using the barrier)



(using barrier)



The threads without the barrier printed out there before and after statements in the order they finished. When running with the barrier however, all the threads stopped once they got to the barrier in the code until all other threads caught up.

In the workerMaster program, I created a program and ran it with and without the use of #pragma omp parallel.

(workerMaster without pragma)

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

Greetings from the master, # 0  of 1 threads

pi@raspberrypi:~ $ ./masterWorker 4

Greetings from the master, # 0  of 1 threads
```

(workerMaster witH pragma)

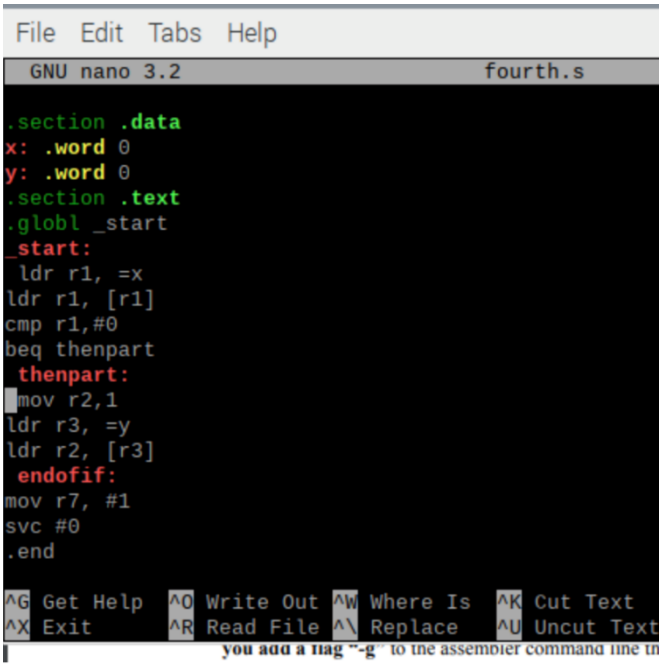
```
pi@raspberrypi:~ $ nano masterWorker.c
pi@raspberrypi:~ $ gcc masterWorker.c -o masterWorker -fopenmp
pi@raspberrypi:~ $ ./masterWorker

Greetings from  a worker, # 1 of 4 threads
Greetings from  a worker, # 2 of 4 threads
Greetings from  a worker, # 3 of 4 threads
Greetings from the master, # 0  of 4 threads

pi@raspberrypi:~ $ ./masterWorker 4

Greetings from  a worker, # 3 of 4 threads
Greetings from the master, # 0  of 4 threads
Greetings from  a worker, # 2 of 4 threads
Greetings from  a worker, # 1 of 4 threads
```

After creating the excutable and running the code, I saw that without the use of parallel, the code would of course only output one line, the master, as the code never splits or forks into separate threads so only thread 0, the master, runs. Once I use the parallel code output the separate threads are created thus allowing for the worker threads to also reply.

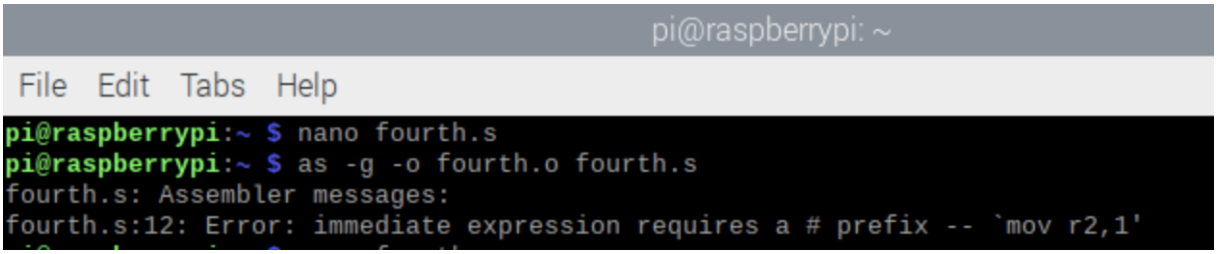
VEDANSI PARSANA

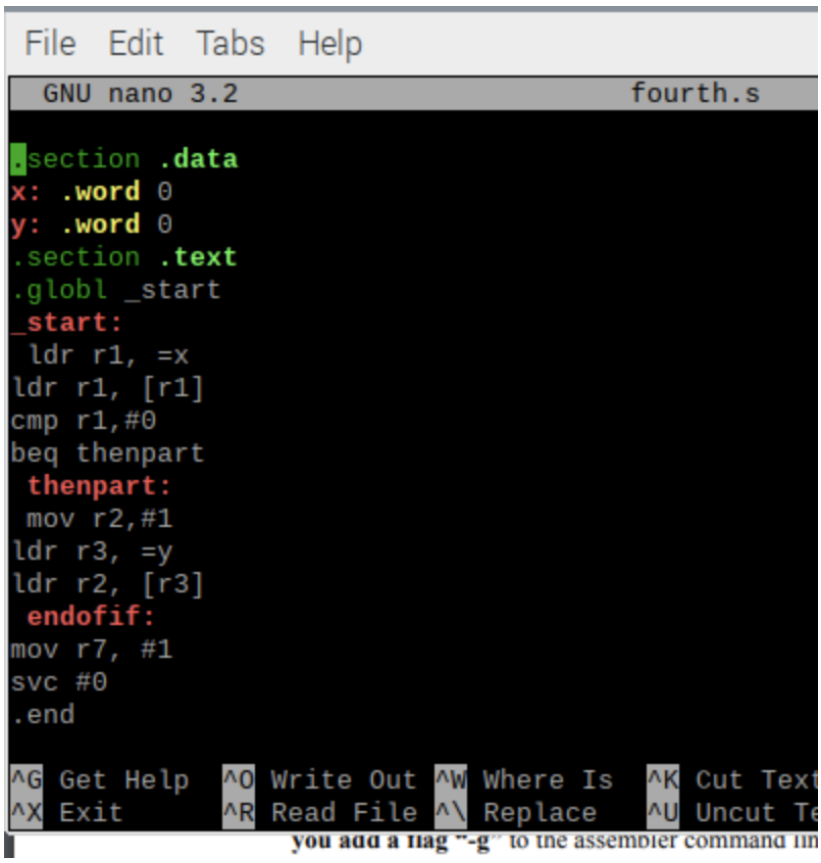
PART 1)

For the fourth program, I read it assembled, linked and executed the file.

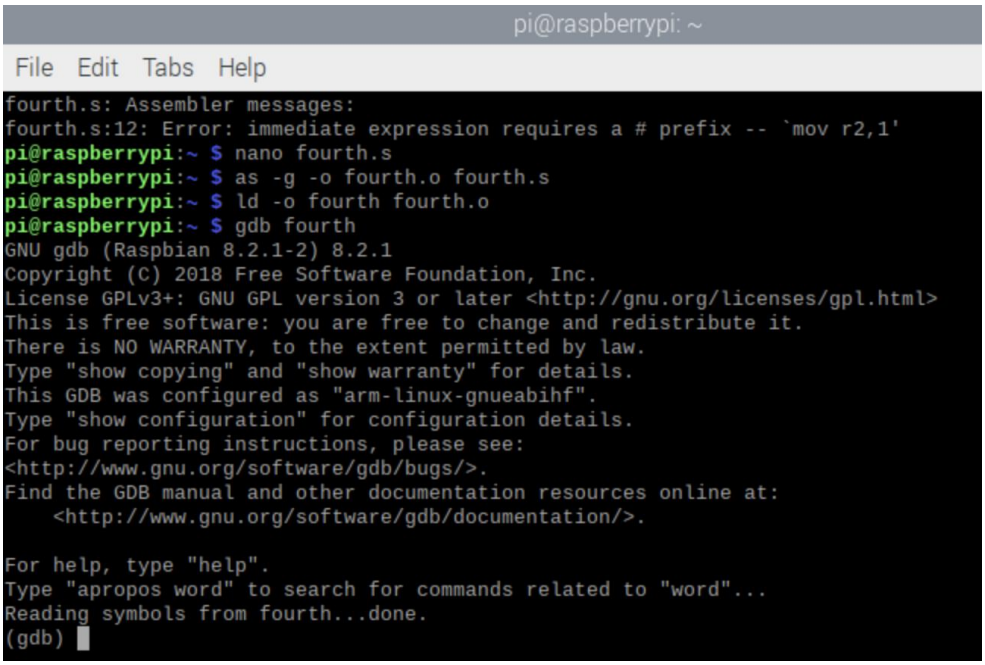


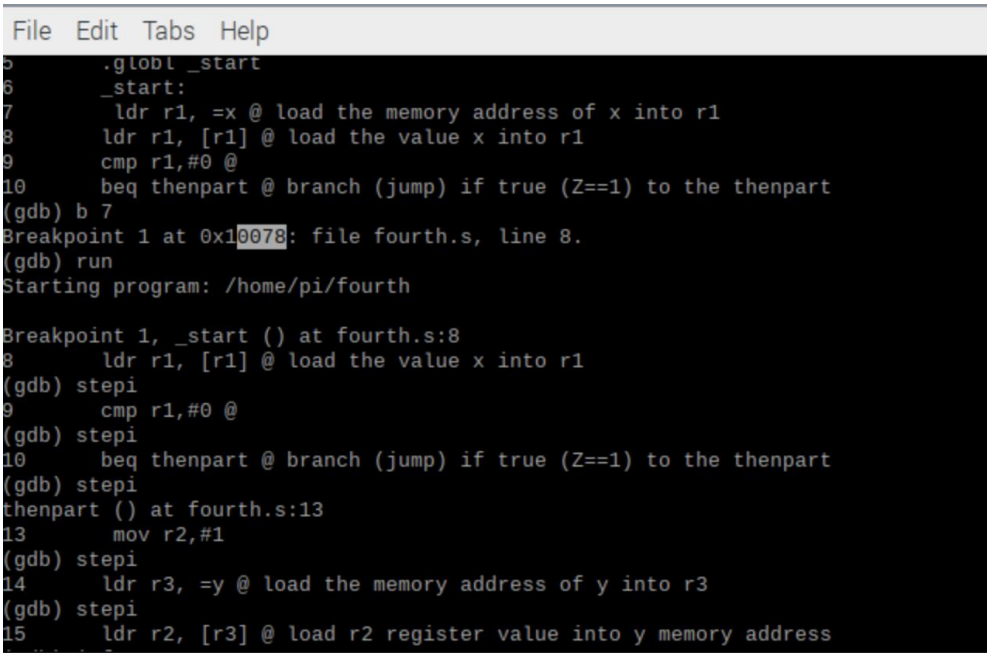
The given code requires a change as the given command ‘mov r2, 1’ is missing a prefix.



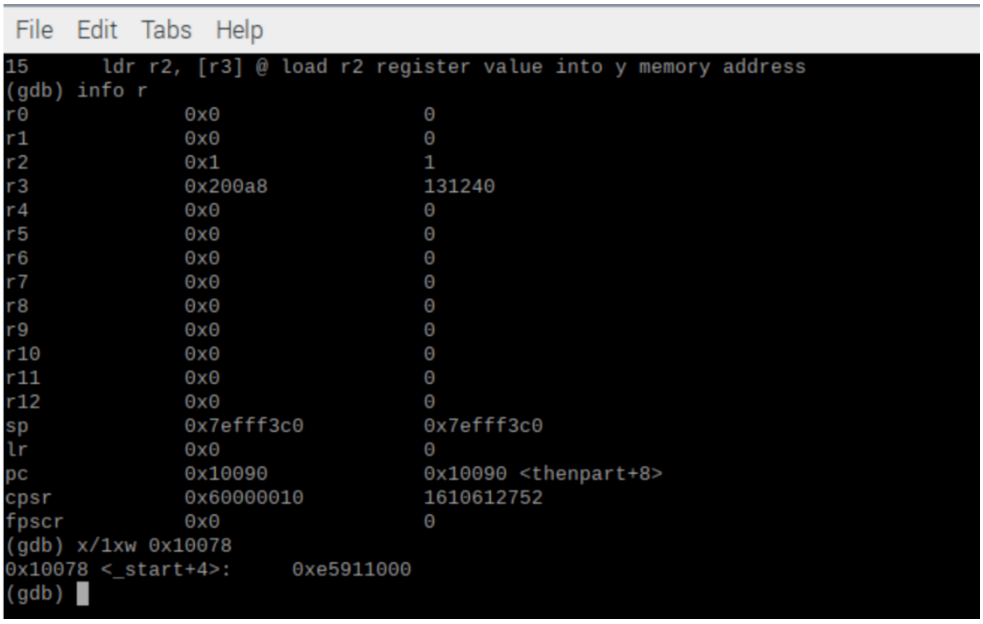


After launching the GNU Debugger, I see the code line numbers using ‘list’ and create a breakpoint at line 7 using ‘b 7’ and test run with ‘run’





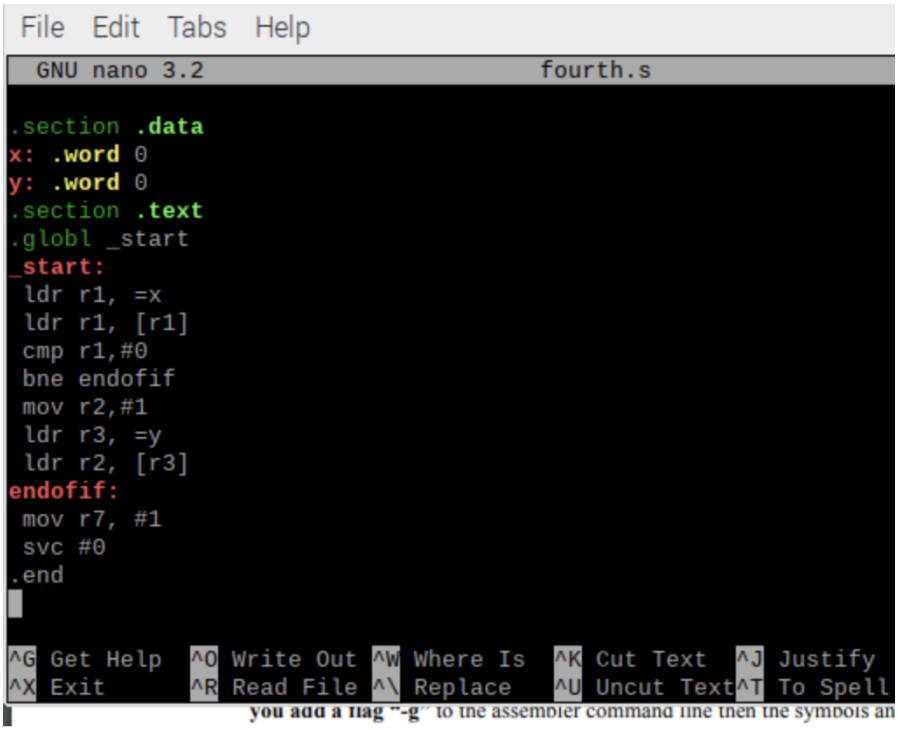
Using ‘stepi’ I continue till before the end, or else it won’t show the registers of the program, then I checked the values of the y memory and the z flag by using ‘x/1xw’ and the given number of our starting point 0x10078, the hexadecimal form 0xe5911000 is where the data is being stored. The register r3, which is holding the y’s address is 0x200a8.



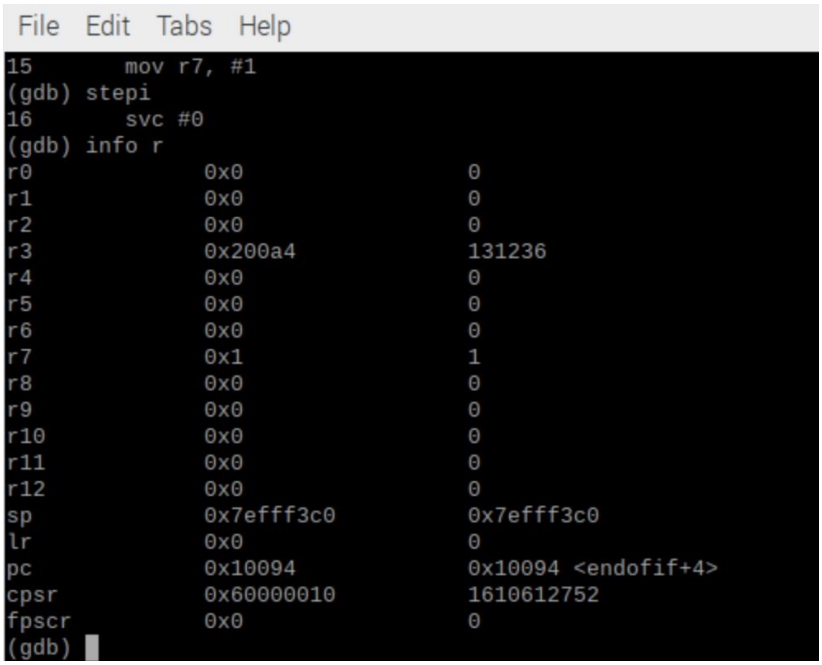
Looking at the registers, we can see that the CPSR is 60000010, meaning the value of the 4 flags (first 4 bits) are 0 1 1 0 to show the flags negative, zero, carry, and 2's complement OV, respectively. Then the zero flag is set as the second bit is one.

PART 2)

For the second part of this task, I updated the fourth.s code so that it only has one branch or jump. Removing the b jump. Also, the bne jump must be set to now jump to the endofif as it is now the opposite condition as beq.



Then I re-assembled, linked and ran it.



the zero flag is still 1 as the CSPR value as the last 4 bits are 0 1 1 0, since x = 0 compares to 0.

PART 3)

Using the fourth program code again, I wrote the given code for the project.

```
 GNU nano 3.2                          ControlStructure1.s

.section .data
x: .word 1
.section .text
.globl _start
_start:
  ldr r1, =x
  ldr r1, [r1]
  ldr r2, =x
  cmp r1, #3
  bgt thenpart
  sub r1, #1
  b endofif
thenpart:
  sub r1, #2
endofif:
  str r1, [r2]
  mov r7, #1
  svc #0

^G Get Help     ^O Write Out    ^W Where Is     ^K Cut Text     ^J
^X Exit         ^R Read File    ^\ Replace      ^U Uncut Text   ^T
```

```
File  Edit  Tabs  Help
17        mov r7, #1
(gdb) stepi
18        svc #0
(gdb) info r
r0             0x0                 0
r1             0x0                 0
r2             0x200a4             131236
r3             0x0                 0
r4             0x0                 0
r5             0x0                 0
r6             0x0                 0
r7             0x1                 1
r8             0x0                 0
r9             0x0                 0
r10            0x0                 0
r11            0x0                 0
r12            0x0                 0
sp             0x7efff3b0          0x7efff3b0
lr             0x0                 0
pc             0x1009c             0x1009c <endofif+8>
cpsr           0x80000010          -2147483632
fpscr          0x0                 0
(gdb)
```

Using the opposite rule, making the condition x > 3, then subtract 2, otherwise continue past to subtract one and unconditional jump to the end. The value 1 of x is set to 0 at the end, with its address in r2 showing as 0x200a4. The z flag is not set as CPSR ends in bits 1 0 0 0. Since x != 3 then the compare wouldn't set off the zero flag.

Appendix

GitHub: https://github.com/hlee113/gitter-geeks/tree/master/A4

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

YouTube: https://www.youtube.com/watch?v=-W_ugGb5dkw