Developing Soft and Parallel Programming Skills Using Project-Based Learning

Semester: Spring 2020

Group Name: CodingCrew

Group Members: Ping He
Hayden Kowalchuk
Vera Barnor Agyiri
Mohammad Munsur Rohan

Planning and Scheduling

Assignee Name	Email	Task	Duration(hours)	Dependency	Due Date	Note
Ping He **(Coordinator)**	phe4@student.gsu.edu	Planning/ scheduling tasks, facilitating meeting rooms and turning in the assignment.	Planning & Scheduling: 2 hours Verification of work and submission: 2 hours	Final report of project A4	03/27/20	100%
Hayden Kowalchuk	hkowalchuk1@student.gsu.edu	Managing and setting up github repo; facilitator	Set up the github repo: 1 hours Providing support to team: 3 hours	Github	03/13/20	100%
Vera Barnor Agyiri	vbarnoragyiri1@student.gsu.edu	Created a new to do, in progress, and done columns for project A4. Editing and uploading the video	Recording, editing, and uploading the video: 3 hours	Youtube	03/25/20	100%
Mohammad Munsur (Rohan)	mmunsur1@student.gsu.edu	Technical writing (getting the report ready) as described in the assignment.	Report: 3 hours	Everyone's report for A4	03/27/20	100%

Parallel Programming Skills

Foundation:

Ping He:

Race Condition:

1. What is race condition?
    a. Race condition is the behavior that the output is dependent on the sequence or timing of other uncontrollable events, which typically represents in electronics, software or other systems.

2. Why is race condition difficult to reproduce and debug?
    a. Because its result is uncertain, it depends on the relative time between the interfering threads. It is difficult to reproduce this complex problem when running in debug mode.

3. How can it be fixed? Provide an example from your Project_A3 (see spmd2.c)
    a. It is better to avoid race conditions by careful software design rather than attempting to fix them afterwards. In spmd2.c, the original code created the variables (threads ID) in a global memory, which led to the thread id appearing repeatedly in one output. The solution to fix this is to declare the variables in the part after the fork (a private memory).

4. 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)

    a. Parallel programs contain many patterns, which are useful ways to write code, and most developers use them repeatedly because they work well in practice. These patterns have been documented by developers so that programmers can learn useful ways to organize and write good parallel code. Patterns can be divided into two broad categories: strategies and concurrent execution mechanisms. The parallel algorithm strategies are used to choose which tasks can be performed by multiple processing units that can be executed simultaneously. Some patterns of the implementation strategy contribute to the overall structure of the program, while others involve how to structure data calculated by multiple processing units. For concurrent execution mode, process / thread control patterns dictate how to control the processing units executing in parallel on the hardware at runtime; the coordination patterns set how to perform multiple tasks on the processing unit simultaneously to coordinate to complete the required parallel computing

5. In the section "Categorizing Patterns" in the "Introduction to Parallel Computing_3.pdf" compare the following:

- Collective synchronization (barrier) with Collective communication (reduction)
    a. A barrier defines a point in the code where all active threads will stop until all threads have arrived at that point to ensure that certain calculations are finished. The reduction specifies one or more thread-private variables that are subject to a reduction operation at the end of the parallel region.

- Master-worker with fork join
    a. Master-Worker: A main process breaks a problem into several sub-problems and assigns them to multiple workflows.
    Fork-join is a way to build and execute parallel programs, so that execution will branch in parallel at a specified point in the program, and "merge" and resume sequential execution at subsequent points.

Dependency: Using your own words and explanation, answer the following:
6. Where can we find parallelism in programming ?
  a. We can find parallelism in program (task) view, data view and resource view.

7. What is dependency and what are its types (provide one example for each)?
  a. Dependency is a situation in which an operation refers to the result of a preceding statement.
  b. 1) true (flow) dependence: a=1; b=a
    2) output dependence: a=f(x); a=b
    3) anti-dependence: a=b; b=1

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

  a. If the order of the execution of two statements doesn't have influence on the result, the statements are independent of each other; if the order of the execution affects the output, two statements are dependent.

  b. Examples:

    1) independent: s1: a=1; s2: b=1. The order of the execution of two statements doesn't matter.

    2) dependent: s1: a=1; s2: b=2*a. If s2 execute first, b will be 2*a (default value

9. When can two statements be executed in parallel?
  a. If and only if statement 1 and statement 2 are independent of each other.

10. How can dependency be removed?
  a. By modifying the program with rearranging statements or eliminating statements.

11. How do we compute dependency for the following two loops and what type/s of dependency?
  a. We can unroll the loop into separate iterations and show their dependences between iterations; for these two loops, their statements have no relationship with earlier results, thus both of them are independent.

Hayden Kowalchuk:

Race Condition:

1. What is race condition?
   a. A race condition occurs in software, electronics and systems where an output depends on proper sequencing of events but this sequence can be changed by uncontrollable events or outside forces.

2. Why is race condition difficult to reproduce and debug?
   a. When a race condition manifests itself as a bug it is nondeterministic and thus employing typical measures to trace the source of the issue can lead to situations that change the behavior of the software or system and the race condition may not occur.

3. How can it be fixed? Provide an example from your Project_A3 (see spmd2.c)
   a. This undesired behavior can generally be avoided in the first place by properly taking into account the common pitfalls of multithreaded execution and designing solutions that adhere to best practices. The given spmd2.c had a critical issue where the thread id was stored in global memory of the program but was read and stored from multiple threads, this meant each thread could set a value but then when it needed to read it back some other thread might have changed it. In this example the correct code was to declare the thread id in thread private memory in order to avoid this thrashing.

4. 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)
   a. In the area of Parallel Programming there are common conventions called patterns and these patterns will start to become familiar the more each programmer sees and uses them, additionally using these will make your code more accessible to other people who might be reviewing or revising it. The two major groups of patterns are simply: Strategies, algorithmic and implementation, and Concurrent Execution Mechanisms, Process/Thread control and Coordination. Algorithmic strategies are concerned with finding tasks that can be executed concurrently, and implementation strategies are how you structure your data. Process and Thread control patterns determine how at run time threads and tasks are organized in terms of parallel execution and most are built into the libraries we will be using and hidden from us. The other side of concurrent execution is Coordination patterns which determine how we coordinate multiple tasks to complete a computation.

5. In the section "Categorizing Patterns" in the "Introduction to Parallel Computing_3.pdf" compare the following:
   1. Collective synchronization (barrier) with Collective communication (reduction)
   a. Collective synchronization is a useful pattern when you need all of your threads to reach a certain point before continuing on to the next piece of the task, while a reduction is more aimed toward ensuring a piece of data has been computed by all threads and a final result has been produced. In the case of a reduction we can tell openmp that we have many threads working toward producing a result and that each of those threads needs to

contribute its individual part to the final result with the appropriate mathematical operation.

2. Master-worker with fork join
   a. Master-worker is a means of organizing parallel programs where a single master thread assigns workloads to worker threads at various points and then gathers the computations from each. This is a different method of control than the fork-join pattern where at defined points in the program it splits into many threads all working on the same workload and then rejoins into a single thread once the computation is finished.

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

6. Where can we find parallelism in programming ?
   a. When looking at tasks we can look at the spaces in between and also try to determine which statement can be executed in parallel. At a higher view we can determine if entire loops, routines and processes would benefit from parallel execution. Depending on the structure and order of our data we can also see if this would be a good place to introduce parallelism.

7. What is dependency and what are its types (provide one example for each)?
   a. A dependency is defined as when one operation depends on another previous operation to finish before it can start.
   b. The article lists 3 types of dependencies: true (flow) dependence, output dependence and anti-dependence.
      i. A basic example of true flow dependence, Read After Write (RAW), is having a result that is dependent on an earlier computation such as:
         A = 1
         B = A
      ii. Anti-dependence, Write After Read (WAR), is the situation in which the first operation is dependent on the second and this is an example of it:
         A = 2
         B = A + 1
         A = 0
      iii. Output dependence, Write After Write (WAW), and an example of this would be when 2 things need to be written but the order is important.
         A = B + C
         A = B + D

8. When is a statement dependent and when it is independent(Provide two examples)?
   a. If there are two statements and the order in which they are executed does not change the result and the execution of each statement doesn't interfere with the other then they are said to be independent of each other.
      Example of this: A= 1, B = 2
      These 2 statements do not interfere with each other nor does their order change the outcome

b. Statements are dependent on each other when the order in which they execute does change the result, and there are multiple types of dependency.
   Example of dependent statements: A = 2, B = A*2
   The ordering of these statements does change the outcome.

9. When can two statements be executed in parallel?
   a. They can be executed in parallel if and only if they are independent of each other

10. How can dependency be removed?
   a. These dependencies can be removed by either rearranging statements or eliminating statements.

11. How do we compute dependency for the following two loops and what type/s of dependency?
   a. For each of the loops we can unroll them into separate statements and iterations to show what each loop is doing and then we can use this to show the dependencies, if any, between each iteration. The first loop has no dependencies between iterations because it only has a single statement within, and this statement could be executed in any order as long as all the statements are executed. The second loop does have 2 statements but in a similar situation with the first loop there are no dependencies in it. S1 is setting the nth index in a to the iterator i, and S2 is setting the nth index in b to the iterator doubled, the order of these statements do not affect each other.

Vera Barnor Agyiri:

Race Condition:

1. What is race condition?
    a. Is the behavior of an electronics, software, or other system where the output is dependent on the sequence or timing of other uncontrollable events.

2. Why is race condition difficult to reproduce and debug?
    a. Race condition is difficult to reproduce and debug because the end result is nondeterministic which depends on the relative timing between interfering threads and when a problem occurs in the production system it can disappear in debug mode.

3. How can it be fixed? Provide an example from your Project_A3 (see spmd2.c)
    a. It can be fixed by carefully planning and writing the software design rather than attempting to fix them afterwards. In spmd2.c we had issues with the threads where some of the thread id was appearing more than once. The way we fixed this was by declaring the thread id into a private memory to stop the problem we were having.

4. 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)
    a. Parallel programs contain several patterns, some of which are useful ways to write codes that developers often use because of how well they operate. These patterns have been recorded over time by developers so that new programmers can learn useful ways to organize and write good parallel code. Patterns are divided into two major groups which are Strategies and Concurrent Execution Mechanisms. Algorithmic strategy is mainly concerned with which tasks can be performed simultaneously by several processing units operating at the same time whereas Implementation strategy is used by parallel systems that contribute to the overall layout of the program with the manner in which data is processed by a multiple processing unit. Concurrent Execution Mechanisms are often classified into two sub-categories that are Process/Thread and Coordination patterns. Process/Thread control pattern determines how the parallel execution processing units on the hardware are managed at run time. Coordination pattern defines how many tasks coordinates run simultaneously to complete computation.

5. In the section "Categorizing Patterns" in the "Introduction to Parallel Computing_3.pdf" compare the following:
    - Collective synchronization (barrier) with Collective communication (reduction)
    a. In collective synchronization (barrier) the barrier pattern is used in parallel programs to ensure that all threads complete a parallel code segment before execution continues whereas in collective communication (reduction) the reduction variable needs to be private to each thread as it runs. After each thread is finished, the final sum of its individual sum is computed. It has dependency on what all the threads are doing to compute it.

    - Master-worker with fork join

a. In master-worker, the master executes one block of code when it forks while the rest of the threads, called workers, execute a different block of code when they fork whereas in fork join, fork separately break off at a recursive point into many threads(such as the parent thread and child threads) and when join calls they all synchronize into one result.

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

6. Where can we find parallelism in programming ?
   a. We can find parallelism in the programming view, in the task view we can find it in the statement level between program statements, also in the block level, loop level, the routine level and process level in the larger grained program statements and in the data view and the resource view.

7. What is dependency and what are its types (provide one example for each)?
   a. Dependency arises when one operation depends on an earlier operation to complete and then produce a result before this later operation can be performed. There are three types of dependency which are:
   b. True Dependencies also called Read-After-Write: the second depends on the first.
      Example: S1: a = 1
      S2: b = a

   c. Anti-Dependencies also called Write-After-Read: the first is dependent on second
      Example: S1 : a = b
      S2 : b = 1

   d. Output Dependencies also called Write-After-Write: the second is also depends on the first
      Example: S1: a = f(x)
      S2: a = b

8. When a statement is dependent and when it is independent(Provide two examples)?
   a. A statement is dependent when one operation depends on an earlier operation to complete and produce a result before this later operation can be performed.
      Example: S1: a = 1
      S2: b = a * 2

   b. A statement is independent when the statement execution does not interfere with each other and the computation results are the same.
      Example: S1: a = 1
      S2: b = 1

9. When can two statements be executed in parallel?
   a. Two statements can execute in parallel if and only if there are no dependencies between the two statements.

10. How can dependency be removed?

a. Dependency can be removed by rearranging and eliminating statements.

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

a. We compute the dependency for the following two loops by unrolling the loop into separate statements and iteration to show the dependencies between the iteration. There's no dependencies for the two loops.

<u>Mohammad Munsur (Rohan):</u>

Race Condition:

1. What is race condition?
   a. The behavior of a certain output that depends on a series/timing of uncontrollable events, exhibited in systems such as electronics, software, and etc.
2. Why is race condition difficult to reproduce and debug?
   a. Because it is nondeterministic as a bug, which means that when one attempts to change the behavior of the software/system in order to apply fixes, the race condition may not occur again.
3. How can it be fixed? Provide an example from your Project_A3 (see spmd2.c)
   a. Taking care to avoid common errors that result from multithreaded execution is the best practice. Program spmd2.c's issue was that the thread id was in shared memory of the threads, but the program was run and stored in multiple different threads. So when the thread read back from memory, the value would have possibly been changed by a different thread. Declaring the thread id in the respective unshared memory of each thread fixed the issue.

4. 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)

   a. Parallel programming has certain distinguishable and efficient ways to code developed by programmers over years, called patterns. These patterns are either Strategies, algorithmic and implementation, or Concurrent Execution Mechanisms, Process/Thread control and Coordination. strategies are implemented to determine which programs to assign to which processing unit to execute at the same time. Coordination patterns show how we coordinate multiple tasks. Process/thread control patterns determine how to direct the processing unit involved in executing parallel and how at run time they are organized. Some strategies determine the whole structure of the program, and some other ones determine how to structure data through multiple processing units to be executed concurrently.

5. In the section "Categorizing Patterns" in the "Introduction to Parallel Computing_3.pdf" compare the following:

- Collective synchronization (barrier) with Collective communication (reduction)
  b. Reduction means that the thread- private variables are going to undergo a reduction operation after the end of that parallel region, while a barrier is what serves as a checkpoint for all active threads to reach and stop at until processes are complete.
- Master-worker with fork join
  b. Master-worker is a central process that breaks down problems into smaller parts of itself and assigns them to different threads/etc. Fork join on the other hand is a term for a parallel program's ability to fork away from each other at a specified point and merge back to each other and continue at other respective points.

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

b. Parallelism is found in resource view, data view, and program (task view).

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

c. Dependency refers to when an operation's function is determined by a preceding function.
1. True (flow) dependence: a = 1, b = a
2. Anti-dependence: a = b, b = 1
3. Output dependence: a = f(x), a = b

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

a. Independence is when the statements involved are not affected by the order of each other. Dependence is when the statements involved are affected by the order of each other
Examples:

1) Independent: line 1: a = 1; line 2: b = 2. Order doesn't determine anything

2) Dependent: line 1: a = 4; line 2: b = 4 * a. Line 1 will need to run first , for b to hold a numeric value. Else, b will hold 4 * a in memory.

9. When can two statements be executed in parallel?

b. Line 1 and Line 2 must be independent of each other.

10. How can dependency be removed?

b. Rearranging statements, eliminating certain parts, and modifying the code.

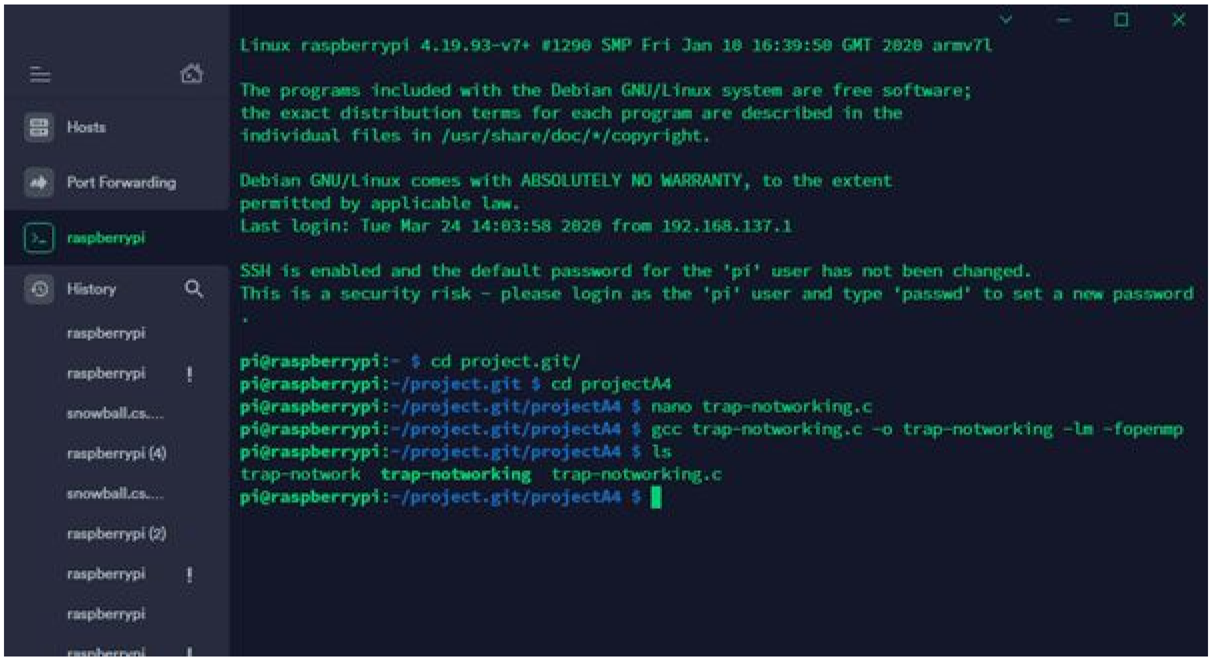
11. How do we compute dependency for the following two loops and what type/s of dependency?

b. We can step through the loops each iteration, and display the relationship to determine the dependency. These loops are independent as they don't affect each other.

Parallel Programming Skills:

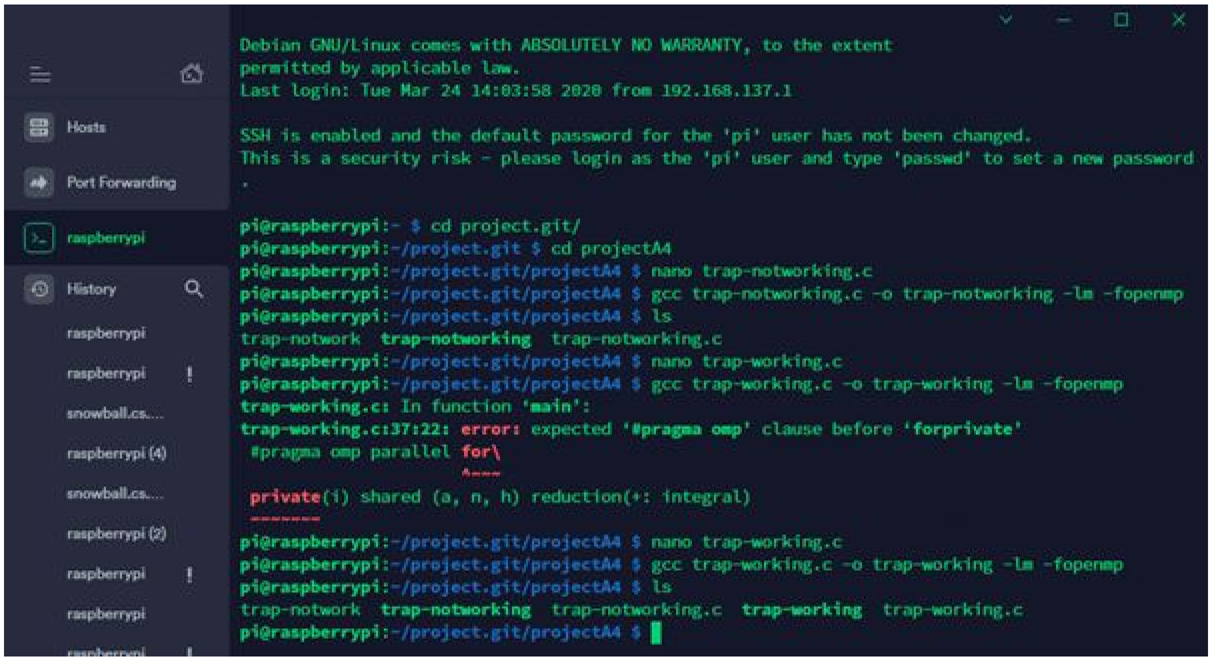
<u>Ping He:</u>

Part1:



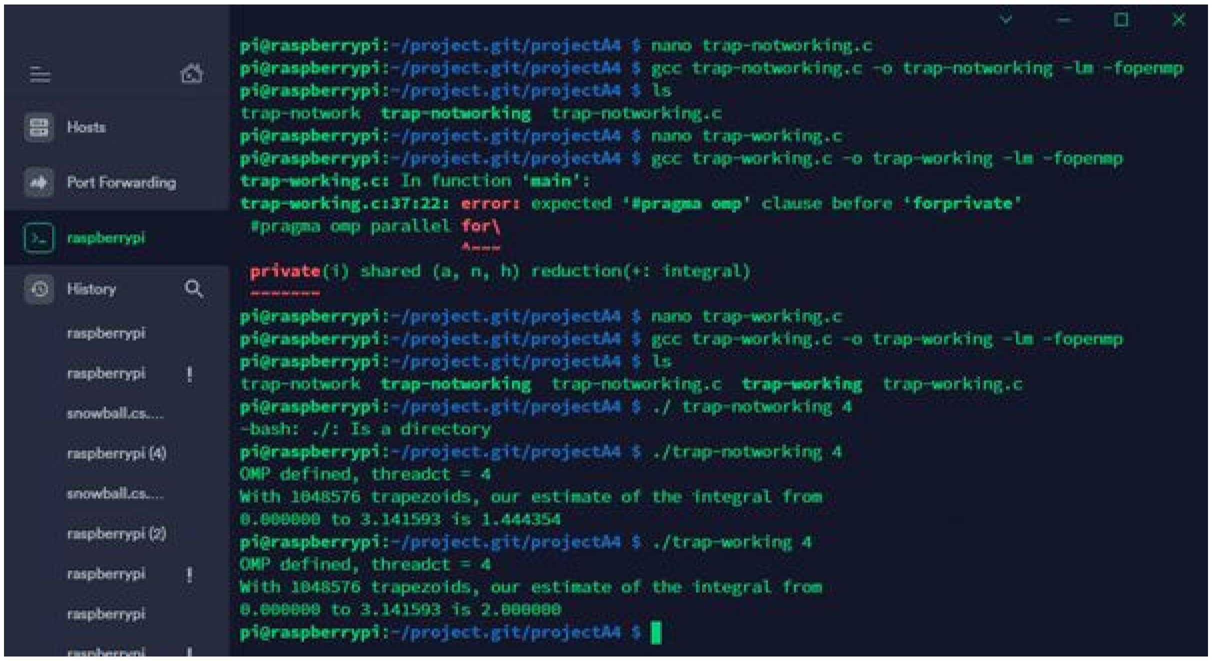
```
Linux raspberrypi 4.19.93-v7+ #1290 SMP Fri Jan 10 16:39:50 GMT 2020 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Mar 24 14:03:58 2020 from 192.168.137.1

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set a new password
.

pi@raspberrypi:~ $ cd project.git/
pi@raspberrypi:~/project.git $ cd projectA4
pi@raspberrypi:~/project.git/projectA4 $ nano trap-notworking.c
pi@raspberrypi:~/project.git/projectA4 $ gcc trap-notworking.c -o trap-notworking -lm -fopenmp
pi@raspberrypi:~/project.git/projectA4 $ ls
trap-notwork  trap-notworking  trap-notworking.c
pi@raspberrypi:~/project.git/projectA4 $
```

The original code has a problem on line 37



```
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Mar 24 14:03:58 2020 from 192.168.137.1

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set a new password
.

pi@raspberrypi:~ $ cd project.git/
pi@raspberrypi:~/project.git $ cd projectA4
pi@raspberrypi:~/project.git/projectA4 $ nano trap-notworking.c
pi@raspberrypi:~/project.git/projectA4 $ gcc trap-notworking.c -o trap-notworking -lm -fopenmp
pi@raspberrypi:~/project.git/projectA4 $ ls
trap-notwork  trap-notworking  trap-notworking.c
pi@raspberrypi:~/project.git/projectA4 $ nano trap-working.c
pi@raspberrypi:~/project.git/projectA4 $ gcc trap-working.c -o trap-working -lm -fopenmp
trap-working.c: In function 'main':
trap-working.c:37:22: error: expected '#pragma omp' clause before 'forprivate'
 #pragma omp parallel for\
                      ^~~~
 private(i) shared (a, n, h) reduction(+: integral)
 ~~~~~~~
pi@raspberrypi:~/project.git/projectA4 $ nano trap-working.c
pi@raspberrypi:~/project.git/projectA4 $ gcc trap-working.c -o trap-working -lm -fopenmp
pi@raspberrypi:~/project.git/projectA4 $ ls
trap-notwork  trap-notworking  trap-notworking.c  trap-working  trap-working.c
pi@raspberrypi:~/project.git/projectA4 $
```

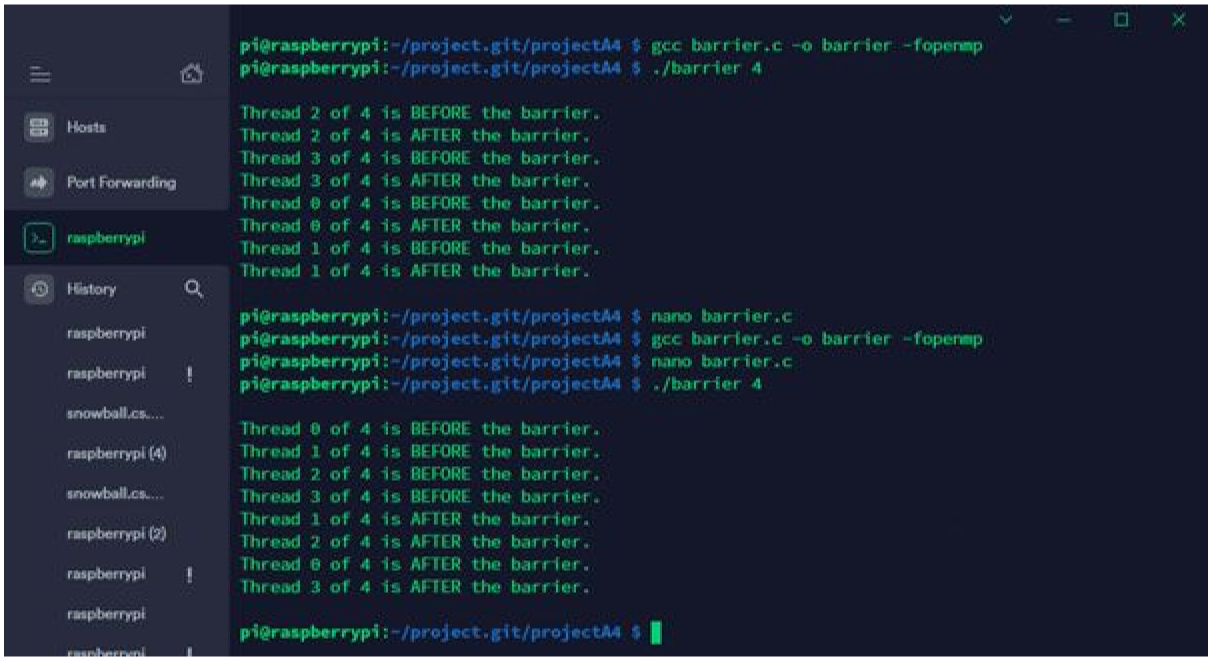
The outputs of improved version:



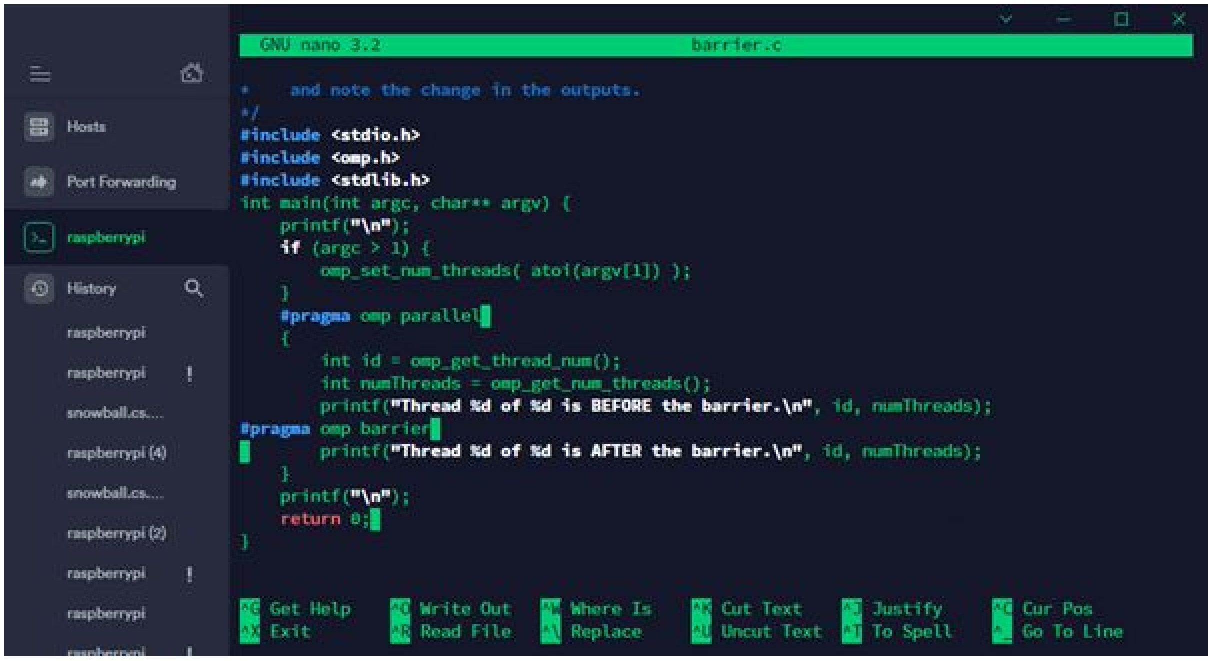
```
pi@raspberrypi:~/project.git/projectA4 $ nano trap-notworking.c
pi@raspberrypi:~/project.git/projectA4 $ gcc trap-notworking.c -o trap-notworking -lm -fopenmp
pi@raspberrypi:~/project.git/projectA4 $ ls
trap-notwork  trap-notworking  trap-notworking.c
pi@raspberrypi:~/project.git/projectA4 $ nano trap-working.c
pi@raspberrypi:~/project.git/projectA4 $ gcc trap-working.c -o trap-working -lm -fopenmp
trap-working.c: In function 'main':
trap-working.c:37:22: error: expected '#pragma omp' clause before 'forprivate'
 #pragma omp parallel for\
                      ^~~~
 private(i) shared (a, n, h) reduction(+: integral)
 ~~~~~~~
pi@raspberrypi:~/project.git/projectA4 $ nano trap-working.c
pi@raspberrypi:~/project.git/projectA4 $ gcc trap-working.c -o trap-working -lm -fopenmp
pi@raspberrypi:~/project.git/projectA4 $ ls
trap-notwork  trap-notworking  trap-notworking.c  trap-working  trap-working.c
pi@raspberrypi:~/project.git/projectA4 $ ./ trap-notworking 4
-bash: ./: Is a directory
pi@raspberrypi:~/project.git/projectA4 $ ./trap-notworking 4
OMP defined, threadct = 4
With 1048576 trapezoids, our estimate of the integral from
0.000000 to 3.141593 is 1.444354
pi@raspberrypi:~/project.git/projectA4 $ ./trap-working 4
OMP defined, threadct = 4
With 1048576 trapezoids, our estimate of the integral from
0.000000 to 3.141593 is 2.000000
pi@raspberrypi:~/project.git/projectA4 $
```

./trap-notworking 4: the result of the integral is wrong, because we didn't add the result of each thread together.
./trap -working 4: the result is right, because we used reduction.

Part2:



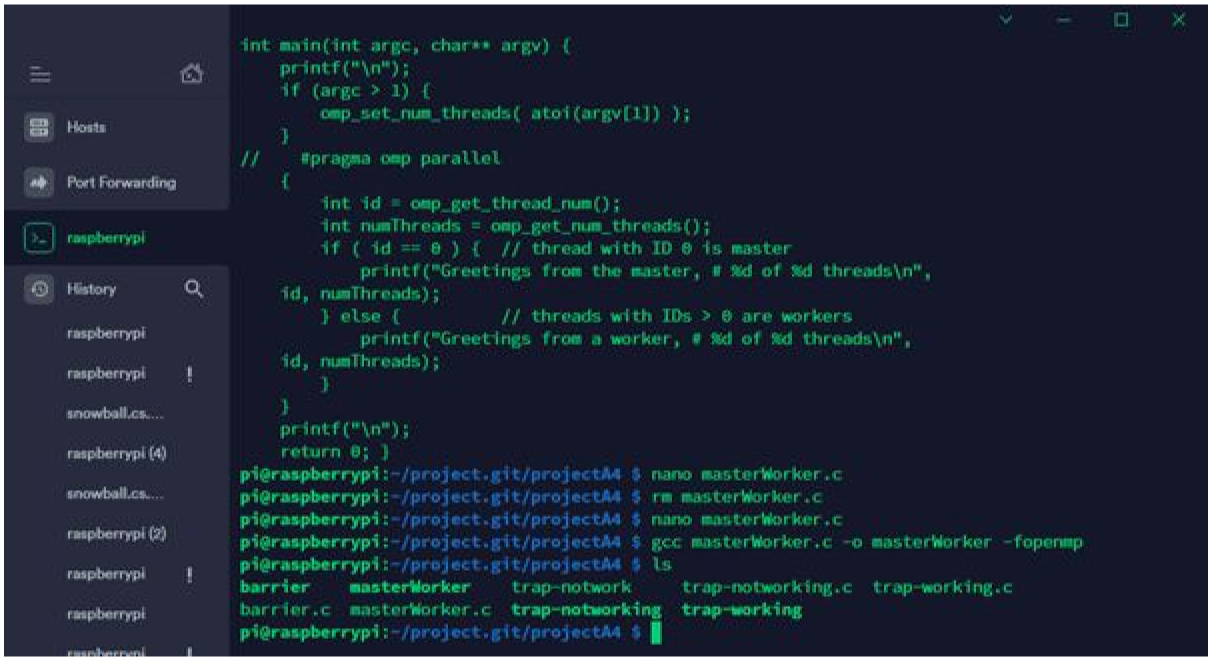
```
pi@raspberrypi:~/project.git/projectA4 $ gcc barrier.c -o barrier -fopenmp
pi@raspberrypi:~/project.git/projectA4 $ ./barrier 4

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

pi@raspberrypi:~/project.git/projectA4 $ nano barrier.c
pi@raspberrypi:~/project.git/projectA4 $ gcc barrier.c -o barrier -fopenmp
pi@raspberrypi:~/project.git/projectA4 $ nano barrier.c
pi@raspberrypi:~/project.git/projectA4 $ ./barrier 4

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

pi@raspberrypi:~/project.git/projectA4 $
```



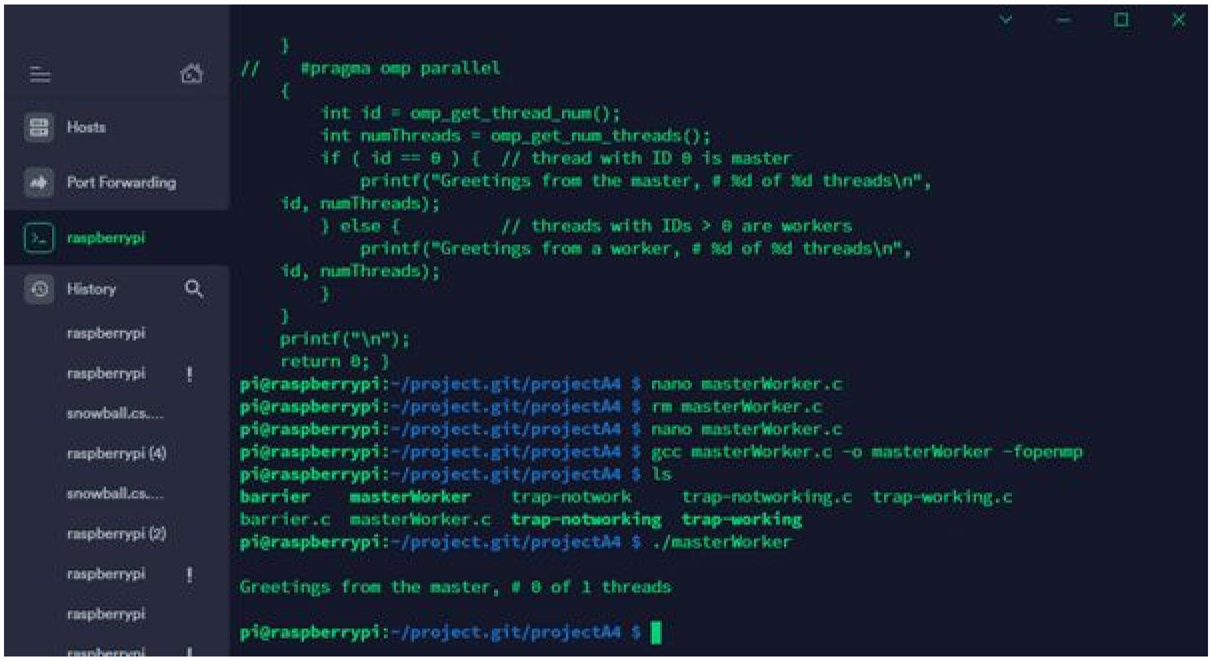
```
  GNU nano 3.2                              barrier.c

*    and note the change in the outputs.
*/
#include <stdio.h>
#include <omp.h>
#include <stdlib.h>
int main(int argc, char** argv) {
    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("Thread %d of %d is BEFORE the barrier.\n", id, numThreads);
#pragma omp barrier
        printf("Thread %d of %d is AFTER the barrier.\n", id, numThreads);
    }
    printf("\n");
    return 0;
}

^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
```

Without the commented pragma: each thread runs to the end.
With the commented pragma: threads stopped until all threads had arrived at the barrier.

Part3:

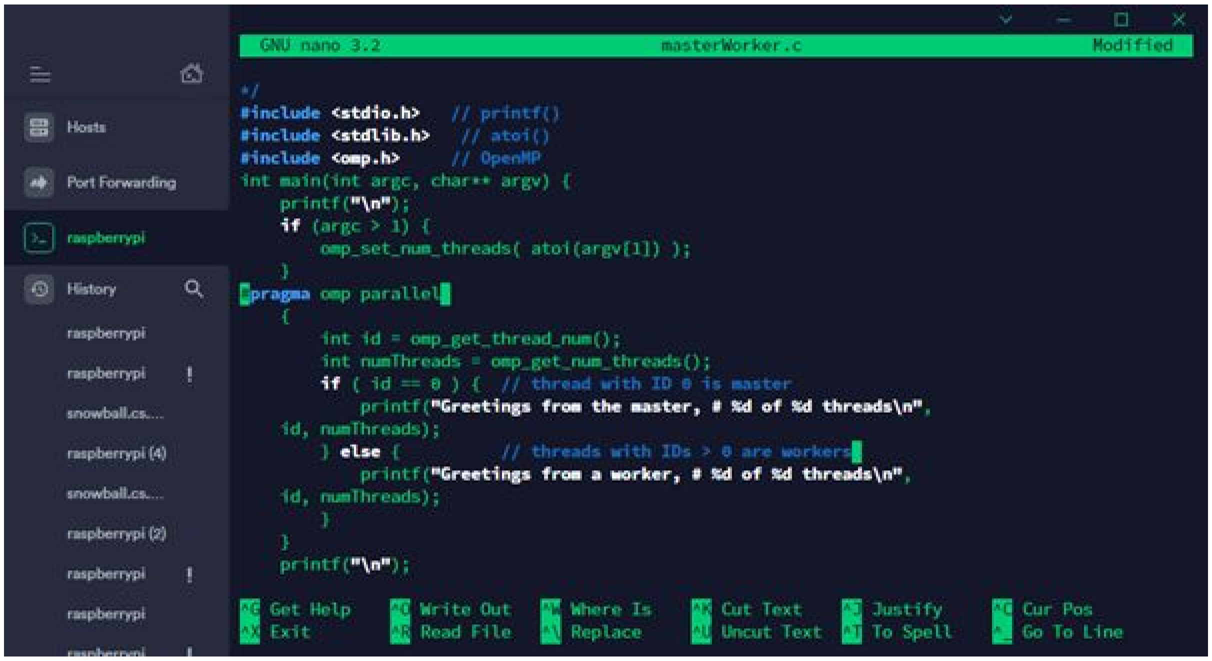


```
int main(int argc, char** argv) {
    printf("\n");
    if (argc > 1) {
        omp_set_num_threads( atoi(argv[1]) );
    }
//    #pragma omp parallel
    {
        int id = omp_get_thread_num();
        int numThreads = omp_get_num_threads();
        if ( id == 0 ) {  // thread with ID 0 is master
            printf("Greetings from the master, # %d of %d threads\n",
    id, numThreads);
        } else {          // threads with IDs > 0 are workers
            printf("Greetings from a worker, # %d of %d threads\n",
    id, numThreads);
        }
    }
    printf("\n");
    return 0; }
pi@raspberrypi:~/project.git/projectA4 $ nano masterWorker.c
pi@raspberrypi:~/project.git/projectA4 $ rm masterWorker.c
pi@raspberrypi:~/project.git/projectA4 $ nano masterWorker.c
pi@raspberrypi:~/project.git/projectA4 $ gcc masterWorker.c -o masterWorker -fopenmp
pi@raspberrypi:~/project.git/projectA4 $ ls
barrier    masterWorker    trap-notwork     trap-notworking.c  trap-working.c
barrier.c  masterWorker.c  trap-notworking  trap-working
pi@raspberrypi:~/project.git/projectA4 $
```



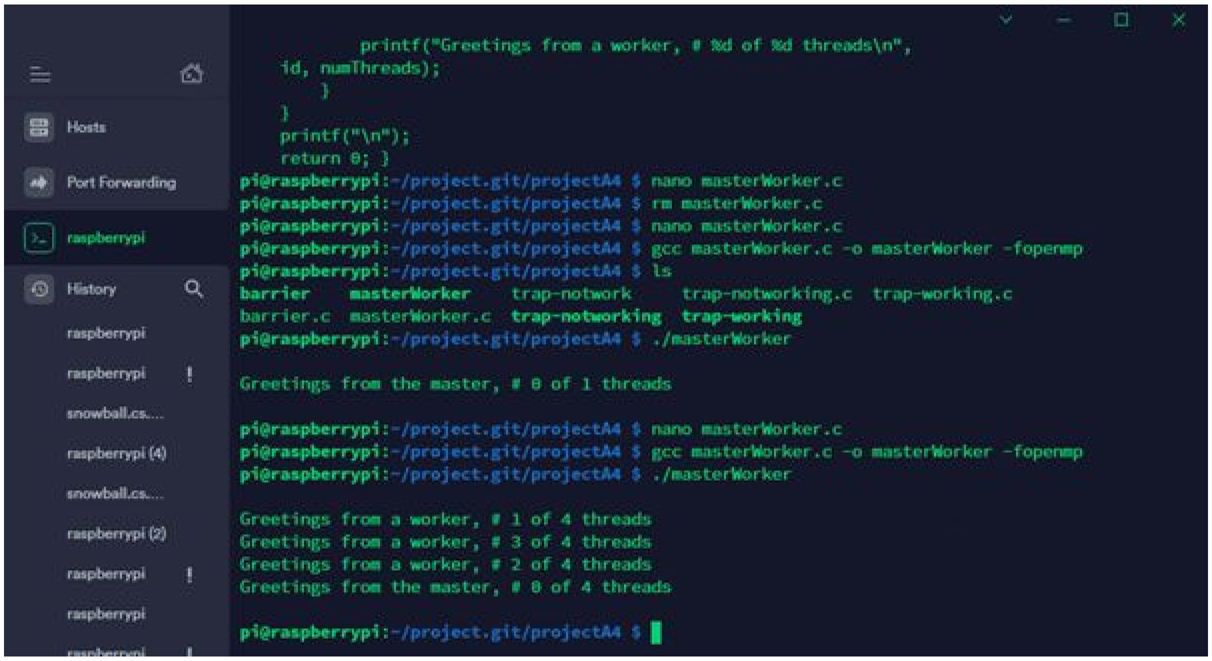
```
    }
//    #pragma omp parallel
    {
        int id = omp_get_thread_num();
        int numThreads = omp_get_num_threads();
        if ( id == 0 ) {  // thread with ID 0 is master
            printf("Greetings from the master, # %d of %d threads\n",
    id, numThreads);
        } else {          // threads with IDs > 0 are workers
            printf("Greetings from a worker, # %d of %d threads\n",
    id, numThreads);
        }
    }
    printf("\n");
    return 0; }
pi@raspberrypi:~/project.git/projectA4 $ nano masterWorker.c
pi@raspberrypi:~/project.git/projectA4 $ rm masterWorker.c
pi@raspberrypi:~/project.git/projectA4 $ nano masterWorker.c
pi@raspberrypi:~/project.git/projectA4 $ gcc masterWorker.c -o masterWorker -fopenmp
pi@raspberrypi:~/project.git/projectA4 $ ls
barrier    masterWorker    trap-notwork     trap-notworking.c  trap-working.c
barrier.c  masterWorker.c  trap-notworking  trap-working
pi@raspberrypi:~/project.git/projectA4 $ ./masterWorker

Greetings from the master, # 0 of 1 threads

pi@raspberrypi:~/project.git/projectA4 $
```

Without the commented pragma: there is only master thread (which ID num is 0) without calling openMp.



```
  GNU nano 3.2                          masterWorker.c                          Modified

*/
#include <stdio.h>   // printf()
#include <stdlib.h>   // atoi()
#include <omp.h>     // OpenMP
int main(int argc, char** argv) {
    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();
        if ( id == 0 ) {  // thread with ID 0 is master
            printf("Greetings from the master, # %d of %d threads\n",
    id, numThreads);
        } else {          // threads with IDs > 0 are workers
            printf("Greetings from a worker, # %d of %d threads\n",
    id, numThreads);
        }
    }
    printf("\n");

^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
```

Uncomment, recompile and execute:



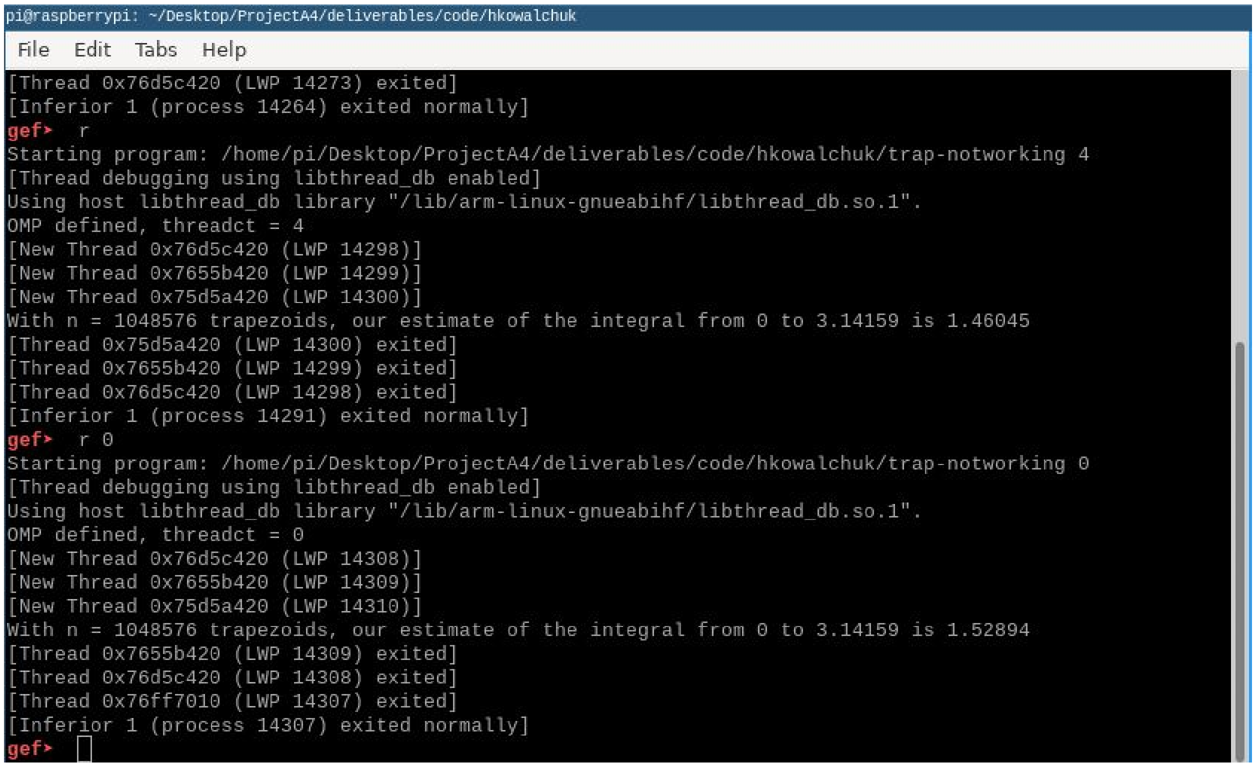
```
            printf("Greetings from a worker, # %d of %d threads\n",
    id, numThreads);
        }
    }
    printf("\n");
    return 0; }
pi@raspberrypi:~/project.git/projectA4 $ nano masterWorker.c
pi@raspberrypi:~/project.git/projectA4 $ rm masterWorker.c
pi@raspberrypi:~/project.git/projectA4 $ nano masterWorker.c
pi@raspberrypi:~/project.git/projectA4 $ gcc masterWorker.c -o masterWorker -fopenmp
pi@raspberrypi:~/project.git/projectA4 $ ls
barrier    masterWorker    trap-notwork     trap-notworking.c  trap-working.c
barrier.c  masterWorker.c  trap-notworking  trap-working
pi@raspberrypi:~/project.git/projectA4 $ ./masterWorker

Greetings from the master, # 0 of 1 threads

pi@raspberrypi:~/project.git/projectA4 $ nano masterWorker.c
pi@raspberrypi:~/project.git/projectA4 $ gcc masterWorker.c -o masterWorker -fopenmp
pi@raspberrypi:~/project.git/projectA4 $ ./masterWorker

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

pi@raspberrypi:~/project.git/projectA4 $
```

With “pragma omp parallel”, we now have 1 master thread and 3 worker thread.

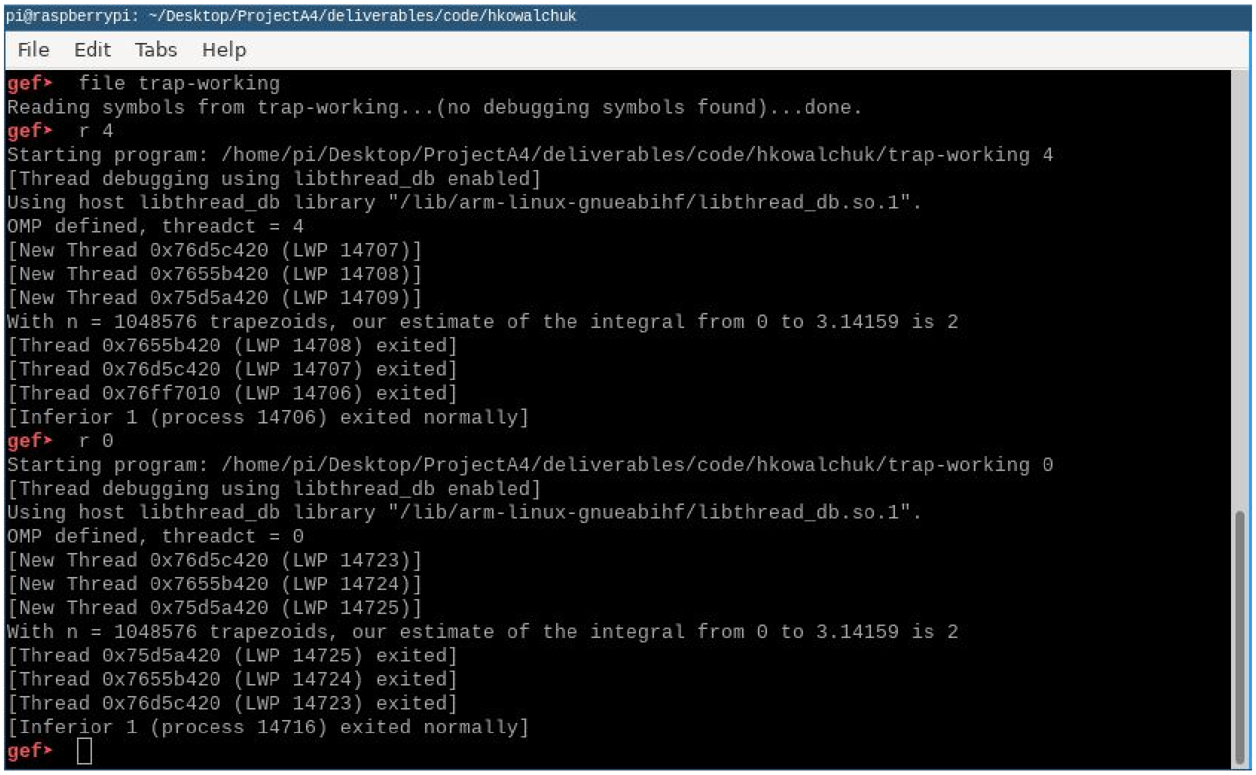
Hayden Kowalchuk:

For the first set of files we were writing and experimenting with, trap-notworking and trap-working, I first transcribed the source for each and then ran with a few different amounts of threads to see any differences. This immediately started to show that the notworking variant was producing wrong results with any amount of threads and that the correct variant using the reduction pragma worked with any number of threads specified. This program although trivial in our case highlights a very important step that care needs to be taken when writing parallel code and how to also correctly identify organizing in the code that may produce incorrect results. For calculating an integral to approximate area the consequences of misculating are very minor, but if this instead were gathering a large dataset for a corporation to then form a 10 year action plan on we could massively off in either direction.

These screenshots show the results from running both versions after compiling them, with the nonworking version first followed by the corrected version:


pi@raspberrypi: ~/Desktop/ProjectA4/deliverables/code/hkowalchuk
File Edit Tabs Help

```
[Thread 0x76d5c420 (LWP 14273) exited]
[Inferior 1 (process 14264) exited normally]
gef➤  r
Starting program: /home/pi/Desktop/ProjectA4/deliverables/code/hkowalchuk/trap-notworking 4
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/arm-linux-gnueabihf/libthread_db.so.1".
OMP defined, threadct = 4
[New Thread 0x76d5c420 (LWP 14298)]
[New Thread 0x7655b420 (LWP 14299)]
[New Thread 0x75d5a420 (LWP 14300)]
With n = 1048576 trapezoids, our estimate of the integral from 0 to 3.14159 is 1.46045
[Thread 0x75d5a420 (LWP 14300) exited]
[Thread 0x7655b420 (LWP 14299) exited]
[Thread 0x76d5c420 (LWP 14298) exited]
[Inferior 1 (process 14291) exited normally]
gef➤  r 0
Starting program: /home/pi/Desktop/ProjectA4/deliverables/code/hkowalchuk/trap-notworking 0
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/arm-linux-gnueabihf/libthread_db.so.1".
OMP defined, threadct = 0
[New Thread 0x76d5c420 (LWP 14308)]
[New Thread 0x7655b420 (LWP 14309)]
[New Thread 0x75d5a420 (LWP 14310)]
With n = 1048576 trapezoids, our estimate of the integral from 0 to 3.14159 is 1.52894
[Thread 0x7655b420 (LWP 14309) exited]
[Thread 0x76d5c420 (LWP 14308) exited]
[Thread 0x76ff7010 (LWP 14307) exited]
[Inferior 1 (process 14307) exited normally]
gef➤
```


pi@raspberrypi: ~/Desktop/ProjectA4/deliverables/code/hkowalchuk
File Edit Tabs Help

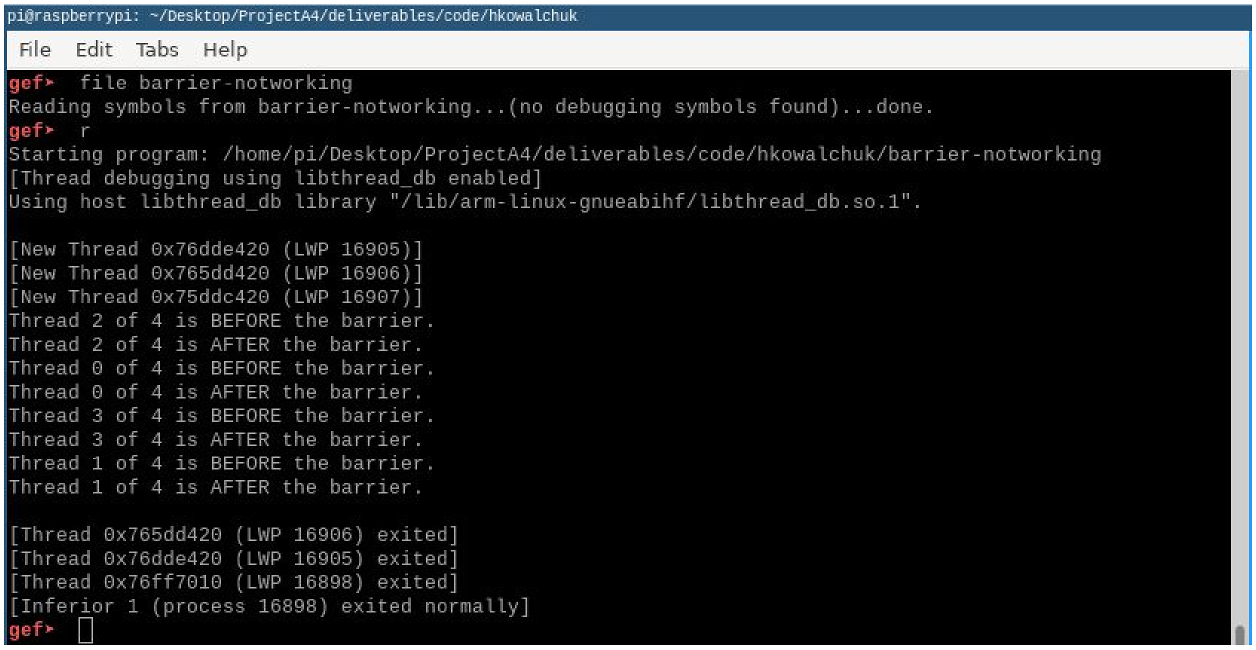
```
gef➤  file trap-working
Reading symbols from trap-working...(no debugging symbols found)...done.
gef➤  r 4
Starting program: /home/pi/Desktop/ProjectA4/deliverables/code/hkowalchuk/trap-working 4
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/arm-linux-gnueabihf/libthread_db.so.1".
OMP defined, threadct = 4
[New Thread 0x76d5c420 (LWP 14707)]
[New Thread 0x7655b420 (LWP 14708)]
[New Thread 0x75d5a420 (LWP 14709)]
With n = 1048576 trapezoids, our estimate of the integral from 0 to 3.14159 is 2
[Thread 0x7655b420 (LWP 14708) exited]
[Thread 0x76d5c420 (LWP 14707) exited]
[Thread 0x76ff7010 (LWP 14706) exited]
[Inferior 1 (process 14706) exited normally]
gef➤  r 0
Starting program: /home/pi/Desktop/ProjectA4/deliverables/code/hkowalchuk/trap-working 0
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/arm-linux-gnueabihf/libthread_db.so.1".
OMP defined, threadct = 0
[New Thread 0x76d5c420 (LWP 14723)]
[New Thread 0x7655b420 (LWP 14724)]
[New Thread 0x75d5a420 (LWP 14725)]
With n = 1048576 trapezoids, our estimate of the integral from 0 to 3.14159 is 2
[Thread 0x75d5a420 (LWP 14725) exited]
[Thread 0x7655b420 (LWP 14724) exited]
[Thread 0x76d5c420 (LWP 14723) exited]
[Inferior 1 (process 14716) exited normally]
gef➤  
```

Code:

```c
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* Demo program for OpenMP: computes trapezoidal approximation to an
integral*/
const double pi = 3.141592653589793238462643383079;
double f(double x);

int main(int argc, char** argv) {
  /* Variables */
  double a = 0.0, b = pi; /* limits of integration */

  int n = 1048576;          /* number of subdivisions = 2^20 */
  double h = (b - a) / n; /* width of subdivision */
  double integral;          /* accumulates answer */
  long threadct = 1;        /* number of threads to use */
  int i;                    /* loop control */

  /* parse command-line arg for number of threads */
  if (argc > 1)
    threadct = strtol(argv[1], NULL, 10);
#ifdef _OPENMP
  printf("OMP defined, threadct = %ld\n", threadct);
#else
  printf("OMP not defined\n");
#endif

  integral = (f(a) + f(b)) / 2.0; /* initialize the variable integral */

#pragma omp parallel for private(i) shared(a, n, h) reduction(+ \
                                                               : integral)
  for (i = 1; i < n; i++) {
    integral += f(a + i * h);
  }

  integral = integral * h;
  printf("With n = %d trapezoids, our estimate ", n);
  printf("of the integral from %g to %g is %g\n", a, b, integral);
}

double f(double x) {
  return sin(x);
}
```

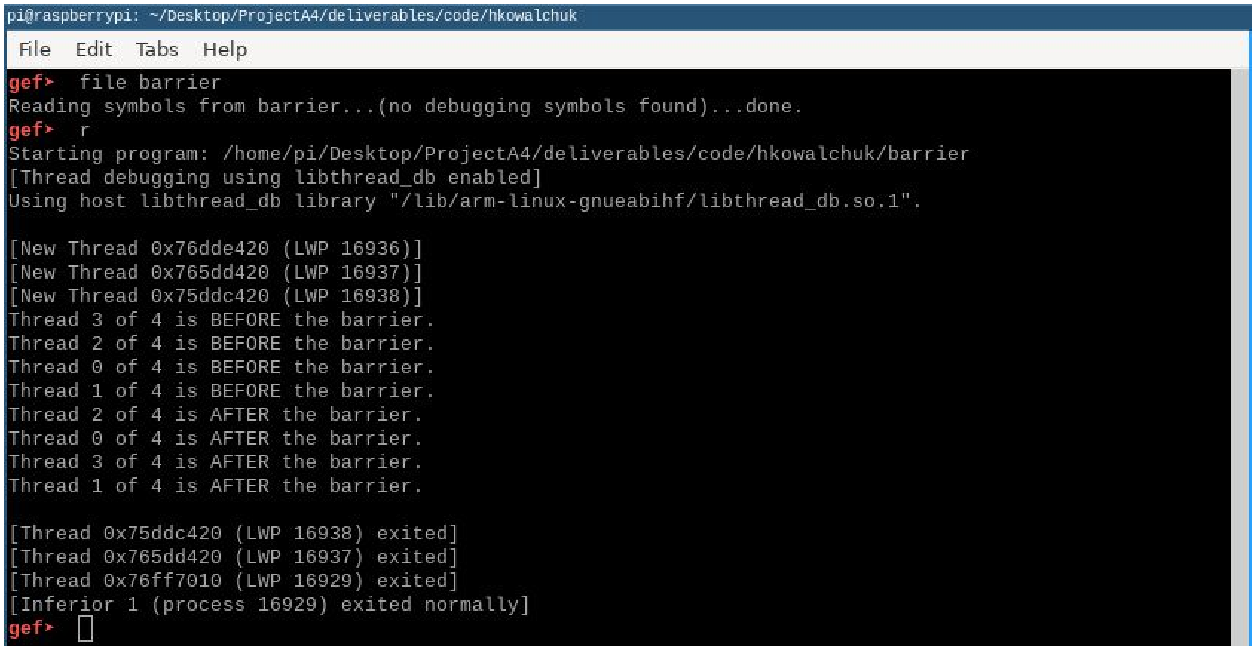
The next task was to examine and learn about control flow barriers, which demarcate an area in the code that all threads must complete up to but not go beyond until all are completed. This ensures that all calculations or operations have completed and data is in place where the programmer would expect it to be. In the basic example we are given the only task each thread has is to print a message containing its thread ID and if it is before or after the barrier. As expected without the barrier each thread prints its before message then immediately prints the after message as well, there is nothing stopping the execution of each thread. Since each thread is free to run its own iteration of the loop the output alternates between before, after, before, after, etc… for all iterations. Once the barrier pragma is implemented in the code, then we see a very different pattern where each thread prints its before message, and after the last thread prints then we see each thread print its after message, thus sorting the messages into 2 distinct blocks.

These images show the contrast between each program and then following the code used:



pi@raspberrypi: ~/Desktop/ProjectA4/deliverables/code/hkowalchuk
File Edit Tabs Help

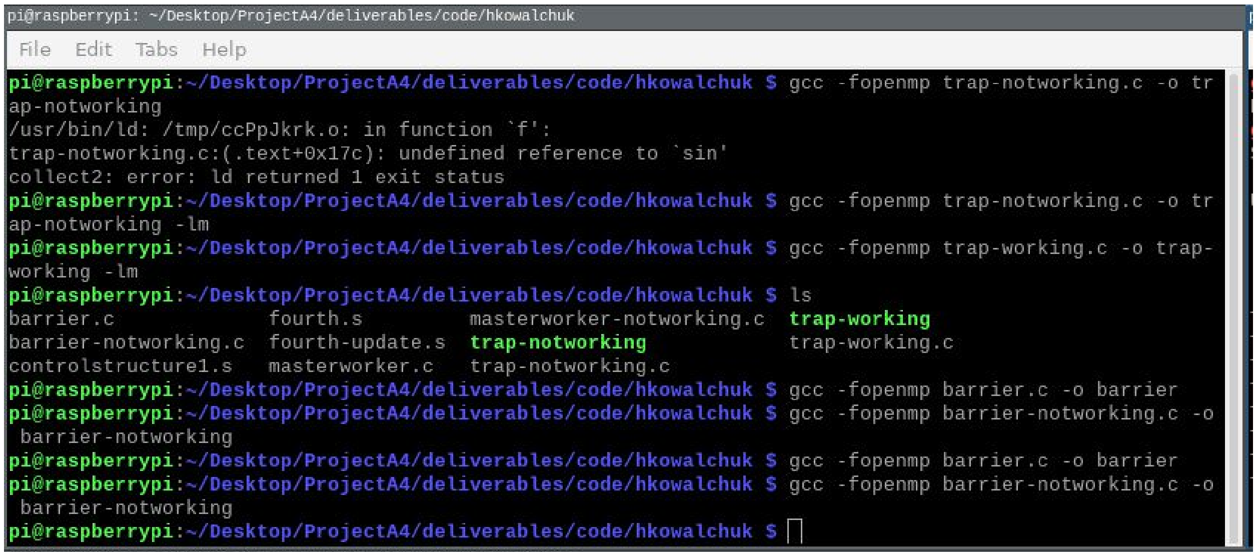
```
gef➤  file barrier-notworking
Reading symbols from barrier-notworking...(no debugging symbols found)...done.
gef➤  r
Starting program: /home/pi/Desktop/ProjectA4/deliverables/code/hkowalchuk/barrier-notworking
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/arm-linux-gnueabihf/libthread_db.so.1".

[New Thread 0x76dde420 (LWP 16905)]
[New Thread 0x765dd420 (LWP 16906)]
[New Thread 0x75ddc420 (LWP 16907)]
Thread 2 of 4 is BEFORE the barrier.
Thread 2 of 4 is AFTER the barrier.
Thread 0 of 4 is BEFORE the barrier.
Thread 0 of 4 is AFTER the barrier.
Thread 3 of 4 is BEFORE the barrier.
Thread 3 of 4 is AFTER the barrier.
Thread 1 of 4 is BEFORE the barrier.
Thread 1 of 4 is AFTER the barrier.

[Thread 0x765dd420 (LWP 16906) exited]
[Thread 0x76dde420 (LWP 16905) exited]
[Thread 0x76ff7010 (LWP 16898) exited]
[Inferior 1 (process 16898) exited normally]
gef➤
```



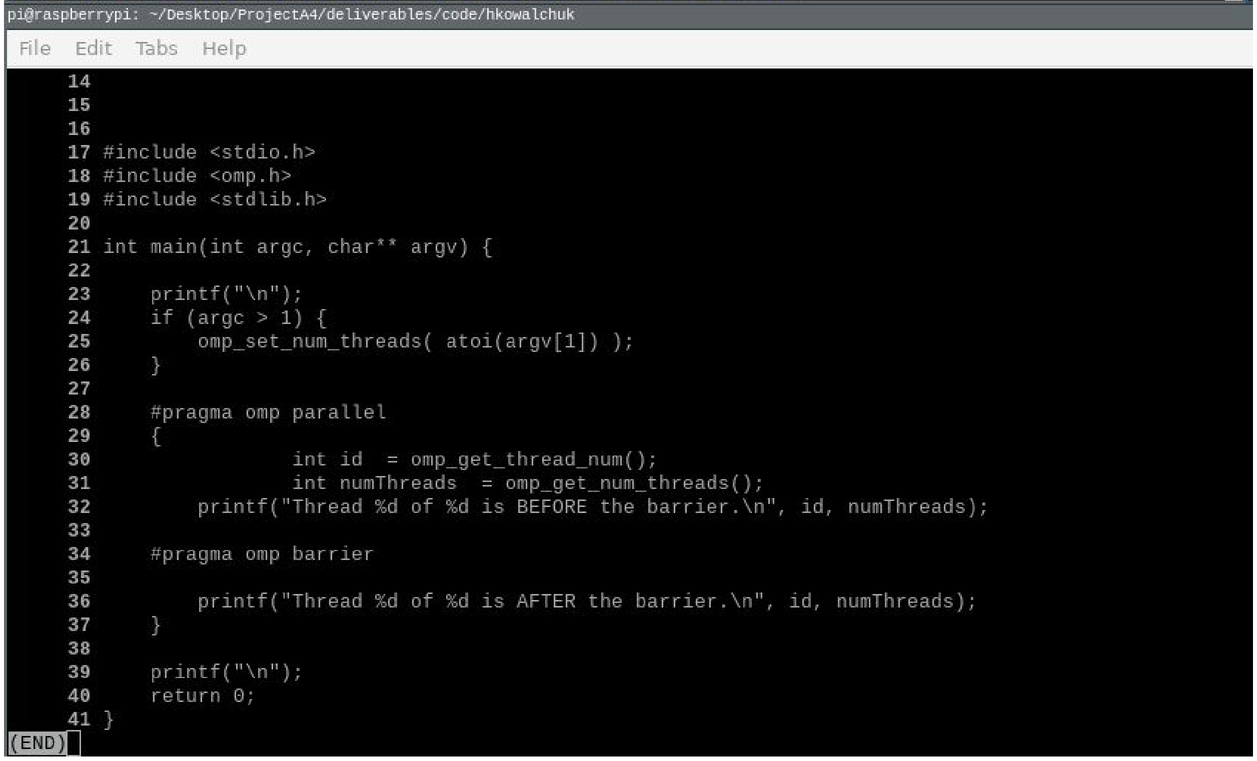
pi@raspberrypi: ~/Desktop/ProjectA4/deliverables/code/hkowalchuk
File Edit Tabs Help

```
gef➤  file barrier
Reading symbols from barrier...(no debugging symbols found)...done.
gef➤  r
Starting program: /home/pi/Desktop/ProjectA4/deliverables/code/hkowalchuk/barrier
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/arm-linux-gnueabihf/libthread_db.so.1".

[New Thread 0x76dde420 (LWP 16936)]
[New Thread 0x765dd420 (LWP 16937)]
[New Thread 0x75ddc420 (LWP 16938)]
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 BEFORE the barrier.
Thread 2 of 4 is AFTER the barrier.
Thread 0 of 4 is AFTER the barrier.
Thread 3 of 4 is AFTER the barrier.
Thread 1 of 4 is AFTER the barrier.

[Thread 0x75ddc420 (LWP 16938) exited]
[Thread 0x765dd420 (LWP 16937) exited]
[Thread 0x76ff7010 (LWP 16929) exited]
[Inferior 1 (process 16929) exited normally]
gef➤
```

The compilation steps and code for barrier.c when working correctly:


pi@raspberrypi: ~/Desktop/ProjectA4/deliverables/code/hkowalchuk
File Edit Tabs Help

```
pi@raspberrypi:~/Desktop/ProjectA4/deliverables/code/hkowalchuk $ gcc -fopenmp trap-notworking.c -o tr
ap-notworking
/usr/bin/ld: /tmp/ccPpJkrk.o: in function `f':
trap-notworking.c:(.text+0x17c): undefined reference to `sin'
collect2: error: ld returned 1 exit status
pi@raspberrypi:~/Desktop/ProjectA4/deliverables/code/hkowalchuk $ gcc -fopenmp trap-notworking.c -o tr
ap-notworking -lm
pi@raspberrypi:~/Desktop/ProjectA4/deliverables/code/hkowalchuk $ gcc -fopenmp trap-working.c -o trap-
working -lm
pi@raspberrypi:~/Desktop/ProjectA4/deliverables/code/hkowalchuk $ ls
barrier.c              fourth.s         masterworker-notworking.c  trap-working
barrier-notworking.c   fourth-update.s  trap-notworking            trap-working.c
controlstructure1.s    masterworker.c   trap-notworking.c
pi@raspberrypi:~/Desktop/ProjectA4/deliverables/code/hkowalchuk $ gcc -fopenmp barrier.c -o barrier
pi@raspberrypi:~/Desktop/ProjectA4/deliverables/code/hkowalchuk $ gcc -fopenmp barrier-notworking.c -o
 barrier-notworking
pi@raspberrypi:~/Desktop/ProjectA4/deliverables/code/hkowalchuk $ gcc -fopenmp barrier.c -o barrier
pi@raspberrypi:~/Desktop/ProjectA4/deliverables/code/hkowalchuk $ gcc -fopenmp barrier-notworking.c -o
 barrier-notworking
pi@raspberrypi:~/Desktop/ProjectA4/deliverables/code/hkowalchuk $ 
```


pi@raspberrypi: ~/Desktop/ProjectA4/deliverables/code/hkowalchuk
File Edit Tabs Help

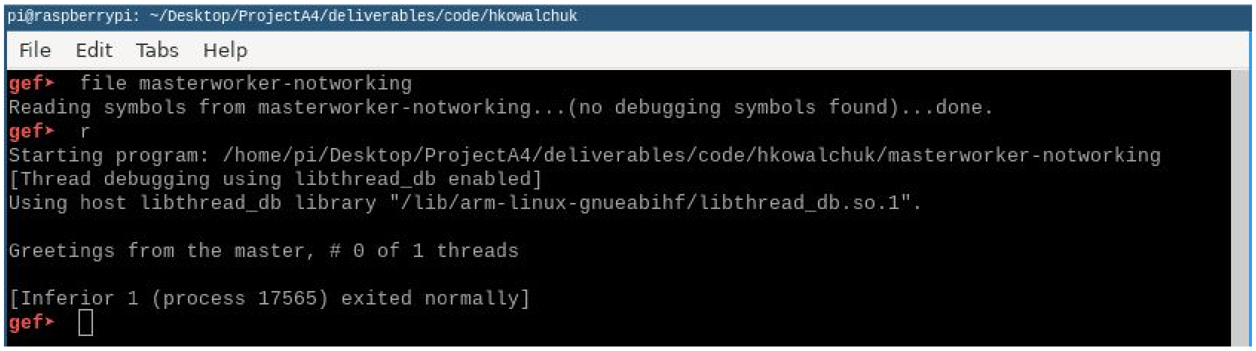
```



#include <stdio.h>
#include <omp.h>
#include <stdlib.h>

int main(int argc, char** argv) {

    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("Thread %d of %d is BEFORE the barrier.\n", id, numThreads);

    #pragma omp barrier

        printf("Thread %d of %d is AFTER the barrier.\n", id, numThreads);
    }

    printf("\n");
    return 0;
}
(END)
```

Code:

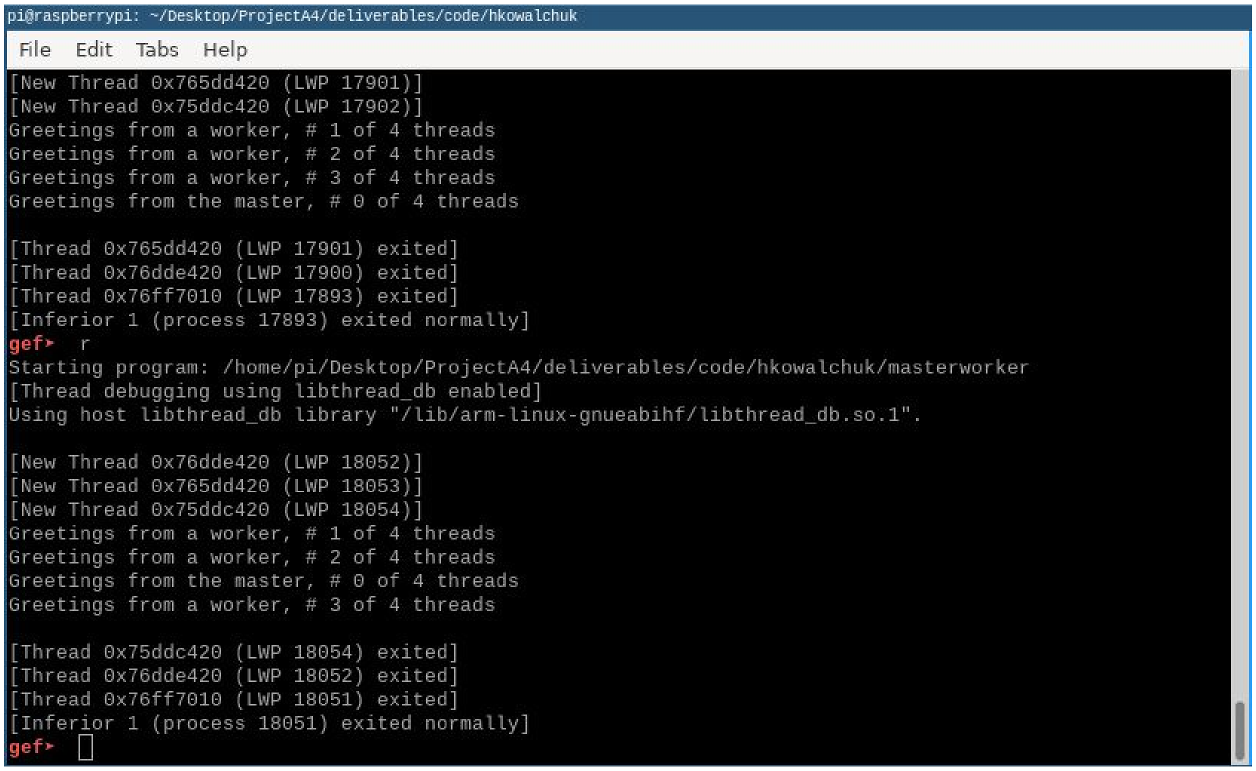
```
/* barrier.c
 * ... illustrates the use of the OpenMP barrier command,
 *  using the commandline to control the number of threads...
 *
 * Joel Adams, Calvin College, May 2013.
 *
 * Usage: ./barrier [numThreads]
 *
 * Exercise:
 * - Compile & run several times, noting interleaving of outputs.
 * - Uncomment the barrier directive, recompile, rerun,
 *   and note the change in the outputs.
 */

#include <stdio.h>
#include <omp.h>
#include <stdlib.h>

int main(int argc, char** argv) {

    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("Thread  %d  of  %d  is  BEFORE  the  barrier.\n",  id,
numThreads);

    #pragma omp barrier

        printf("Thread %d of %d is AFTER the barrier.\n", id, numThreads);
    }

    printf("\n");
    return 0;
}
```

For the final pattern concept we were shown we are using the thread id to determine which thread is the master thread and which other threads are merely workers. Running the code as shown with no parallel pragmas in place just results in a single execution that states it is the master thread, which makes sense because openMP strives to gracefully fallback if the pragmas are not understood or available. In the fixed version with the omp parallel pragma correctly in place we do see that we have 4 threads, 1 master and 3 workers, but that the master thread doesn't necessarily execute first.

These 2 images depict this behavior:



```
pi@raspberrypi: ~/Desktop/ProjectA4/deliverables/code/hkowalchuk
gef➤  file masterworker-notworking
Reading symbols from masterworker-notworking...(no debugging symbols found)...done.
gef➤  r
Starting program: /home/pi/Desktop/ProjectA4/deliverables/code/hkowalchuk/masterworker-notworking
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/arm-linux-gnueabihf/libthread_db.so.1".

Greetings from the master, # 0 of 1 threads

[Inferior 1 (process 17565) exited normally]
gef➤  
```



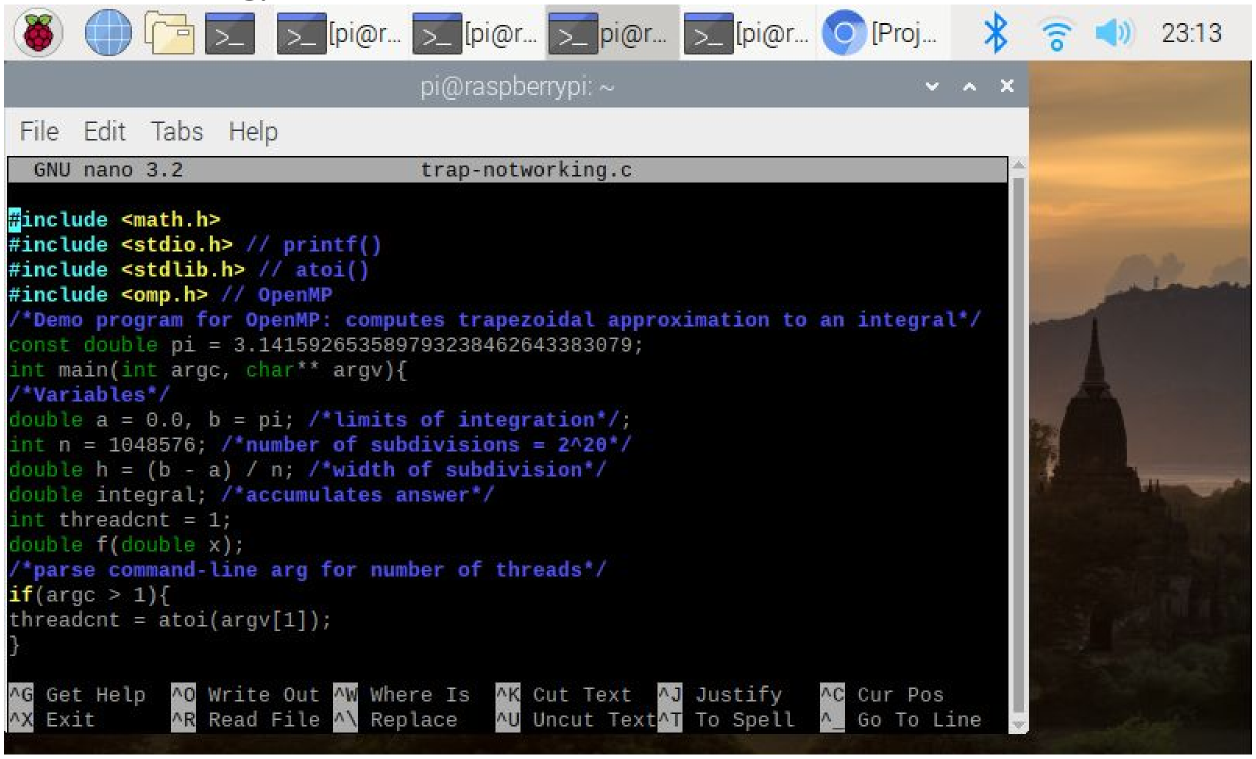
```
pi@raspberrypi: ~/Desktop/ProjectA4/deliverables/code/hkowalchuk
[New Thread 0x765dd420 (LWP 17901)]
[New Thread 0x75ddc420 (LWP 17902)]
Greetings from a worker, # 1 of 4 threads
Greetings from a worker, # 2 of 4 threads
Greetings from a worker, # 3 of 4 threads
Greetings from the master, # 0 of 4 threads

[Thread 0x765dd420 (LWP 17901) exited]
[Thread 0x76dde420 (LWP 17900) exited]
[Thread 0x76ff7010 (LWP 17893) exited]
[Inferior 1 (process 17893) exited normally]
gef➤  r
Starting program: /home/pi/Desktop/ProjectA4/deliverables/code/hkowalchuk/masterworker
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/arm-linux-gnueabihf/libthread_db.so.1".

[New Thread 0x76dde420 (LWP 18052)]
[New Thread 0x765dd420 (LWP 18053)]
[New Thread 0x75ddc420 (LWP 18054)]
Greetings from a worker, # 1 of 4 threads
Greetings from a worker, # 2 of 4 threads
Greetings from the master, # 0 of 4 threads
Greetings from a worker, # 3 of 4 threads

[Thread 0x75ddc420 (LWP 18054) exited]
[Thread 0x76dde420 (LWP 18052) exited]
[Thread 0x76ff7010 (LWP 18051) exited]
[Inferior 1 (process 18051) exited normally]
gef➤  
```

Code:

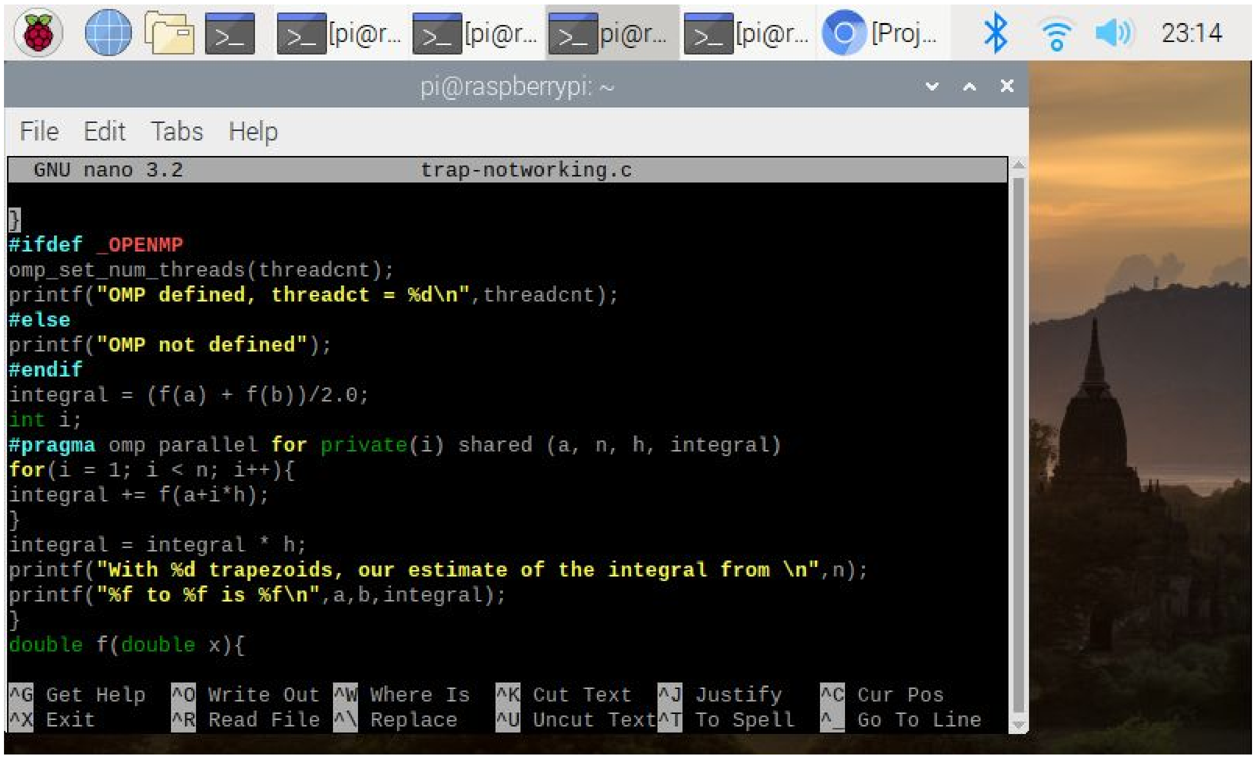
```
/* masterWorker.c
 * ... illustrates the master-worker pattern in OpenMP
 *
 * Joel Adams, Calvin College, November 2009.
 * Usage: ./masterWorker
 * Exercise:
 * - Compile and run as is.
 * - Uncomment #pragma directive, re-compile and re-run
 * - Compare and trace the different executions.
 */

#include <omp.h>     // OpenMP
```

```c
#include <stdio.h>  // printf()

int main(int argc, char** argv) {
  printf("\n");

#pragma omp parallel
  {
    int id = omp_get_thread_num();
    int numThreads = omp_get_num_threads();

    if (id == 0) {  // thread with ID 0 is master
      printf("Greetings from the master, # %d of %d threads\n",
              id, numThreads);
    } else {  // threads with IDs > 0 are workers
      printf("Greetings from a worker, # %d of %d threads\n",
              id, numThreads);
    }
  }

  printf("\n");

  return 0;
}
```

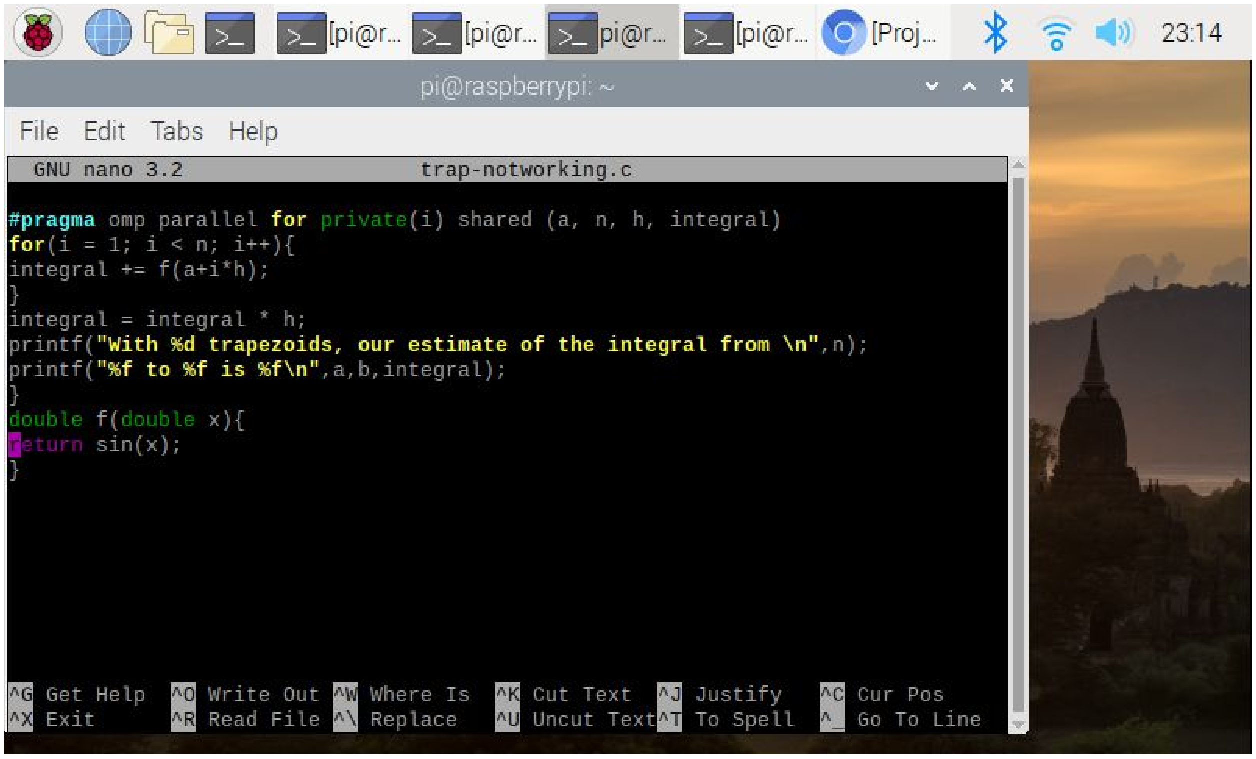
Vera Barnor Agyiri:



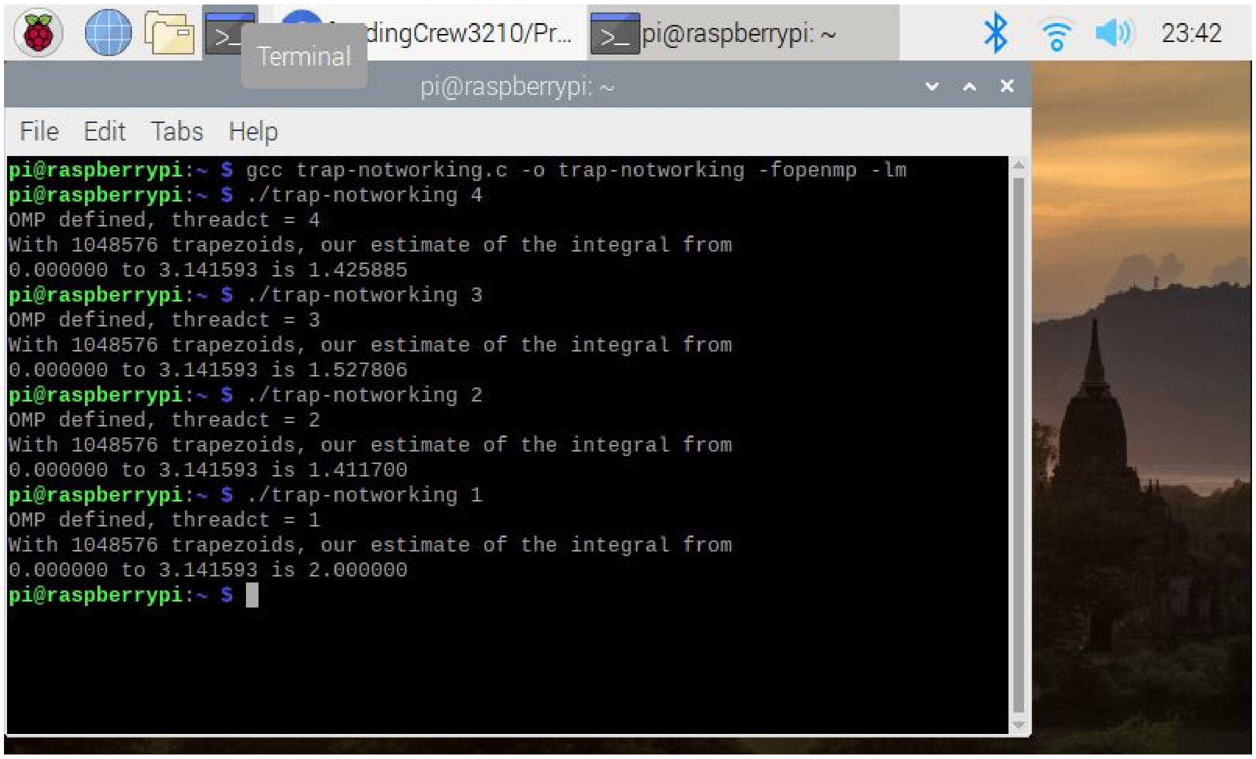
```
GNU nano 3.2                    trap-notworking.c

#include <math.h>
#include <stdio.h> // printf()
#include <stdlib.h> // atoi()
#include <omp.h> // OpenMP
/*Demo program for OpenMP: computes trapezoidal approximation to an integral*/
const double pi = 3.141592653589793238462643383079;
int main(int argc, char** argv){
/*Variables*/
double a = 0.0, b = pi; /*limits of integration*/;
int n = 1048576; /*number of subdivisions = 2^20*/
double h = (b - a) / n; /*width of subdivision*/
double integral; /*accumulates answer*/
int threadcnt = 1;
double f(double x);
/*parse command-line arg for number of threads*/
if(argc > 1){
threadcnt = atoi(argv[1]);
}
```



```
GNU nano 3.2                    trap-notworking.c

}
#ifdef _OPENMP
omp_set_num_threads(threadcnt);
printf("OMP defined, threadct = %d\n",threadcnt);
#else
printf("OMP not defined");
#endif
integral = (f(a) + f(b))/2.0;
int i;
#pragma omp parallel for private(i) shared (a, n, h, integral)
for(i = 1; i < n; i++){
integral += f(a+i*h);
}
integral = integral * h;
printf("With %d trapezoids, our estimate of the integral from \n",n);
printf("%f to %f is %f\n",a,b,integral);
}
double f(double x){
```

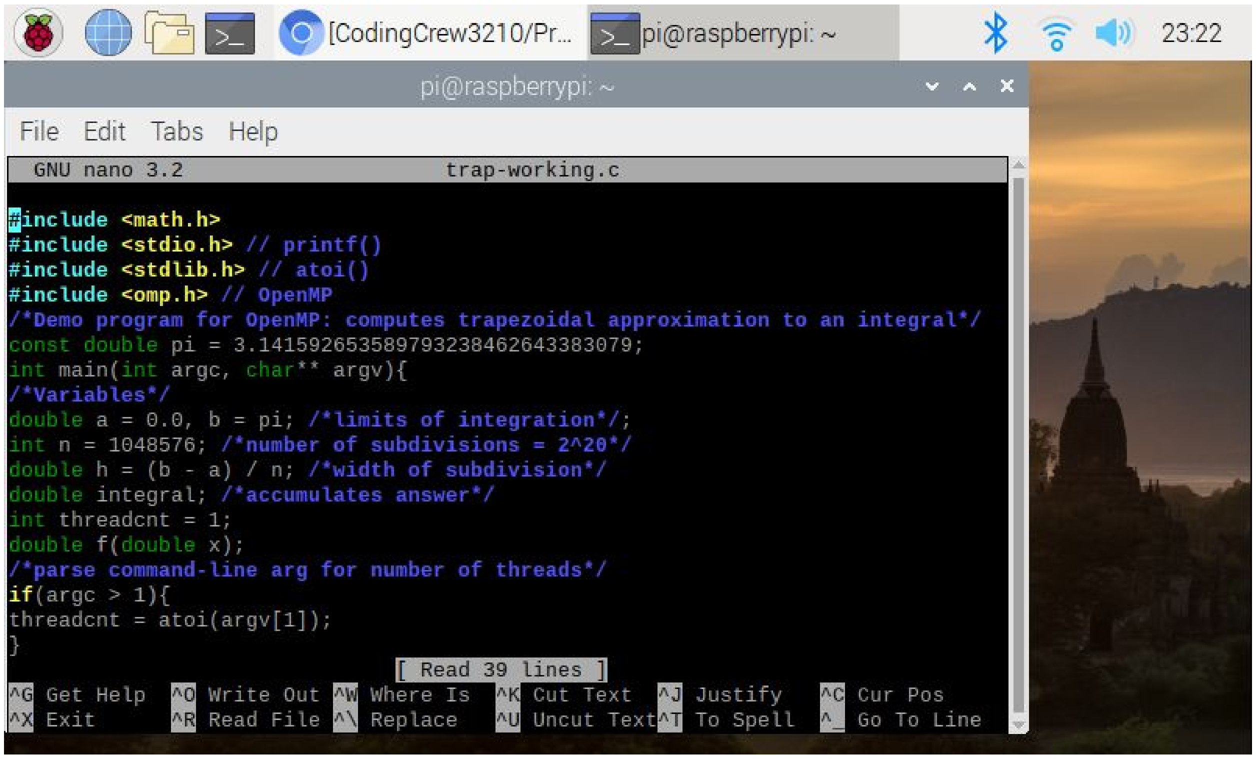


```
 GNU nano 3.2                     trap-notworking.c

#pragma omp parallel for private(i) shared (a, n, h, integral)
for(i = 1; i < n; i++){
integral += f(a+i*h);
}
integral = integral * h;
printf("With %d trapezoids, our estimate of the integral from \n",n);
printf("%f to %f is %f\n",a,b,integral);
}
double f(double x){
return sin(x);
}

^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
```

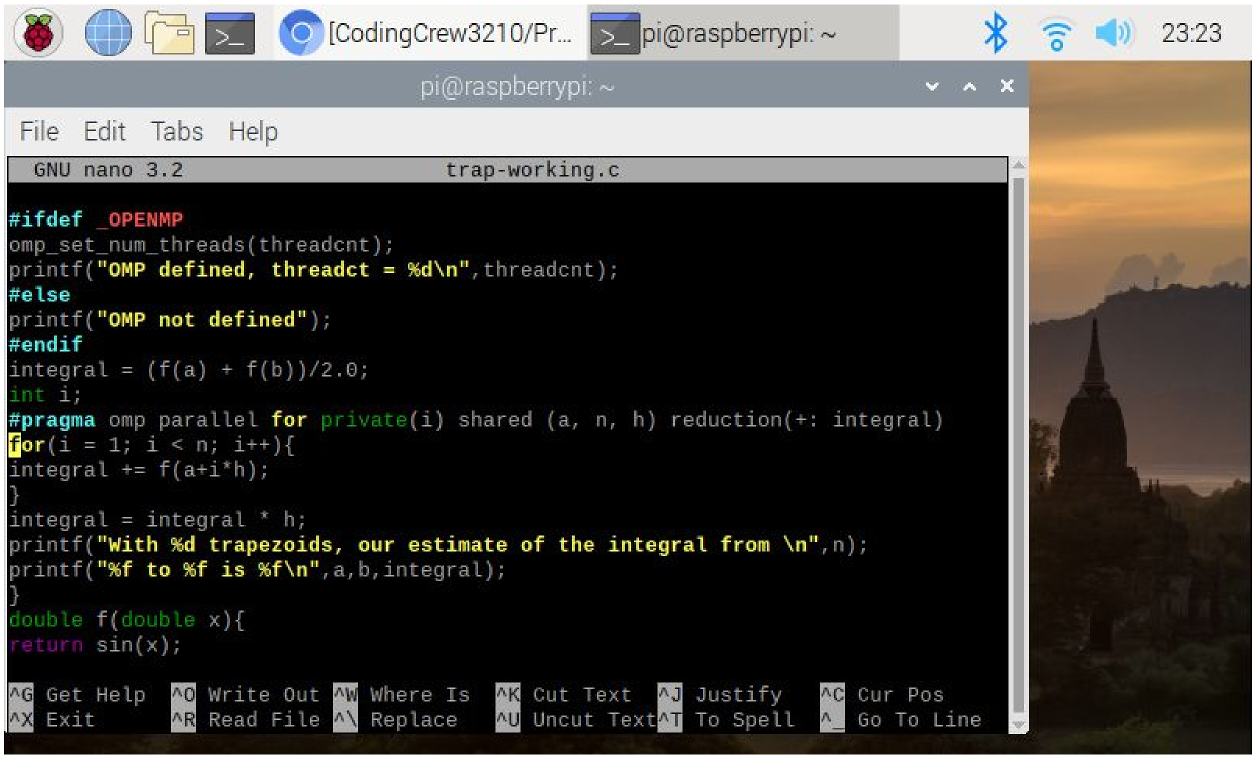


```
pi@raspberrypi:~ $ gcc trap-notworking.c -o trap-notworking -fopenmp -lm
pi@raspberrypi:~ $ ./trap-notworking 4
OMP defined, threadct = 4
With 1048576 trapezoids, our estimate of the integral from
0.000000 to 3.141593 is 1.425885
pi@raspberrypi:~ $ ./trap-notworking 3
OMP defined, threadct = 3
With 1048576 trapezoids, our estimate of the integral from
0.000000 to 3.141593 is 1.527806
pi@raspberrypi:~ $ ./trap-notworking 2
OMP defined, threadct = 2
With 1048576 trapezoids, our estimate of the integral from
0.000000 to 3.141593 is 1.411700
pi@raspberrypi:~ $ ./trap-notworking 1
OMP defined, threadct = 1
With 1048576 trapezoids, our estimate of the integral from
0.000000 to 3.141593 is 2.000000
pi@raspberrypi:~ $
```

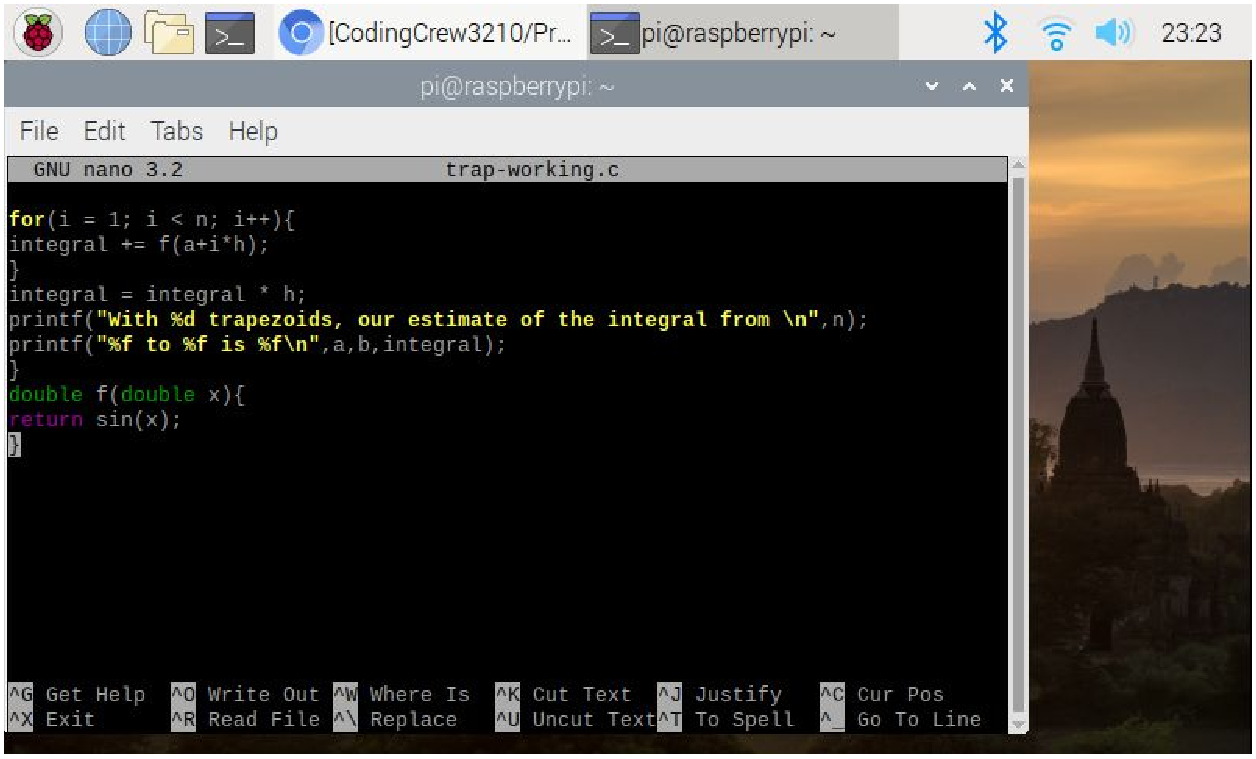
For trap-notworking



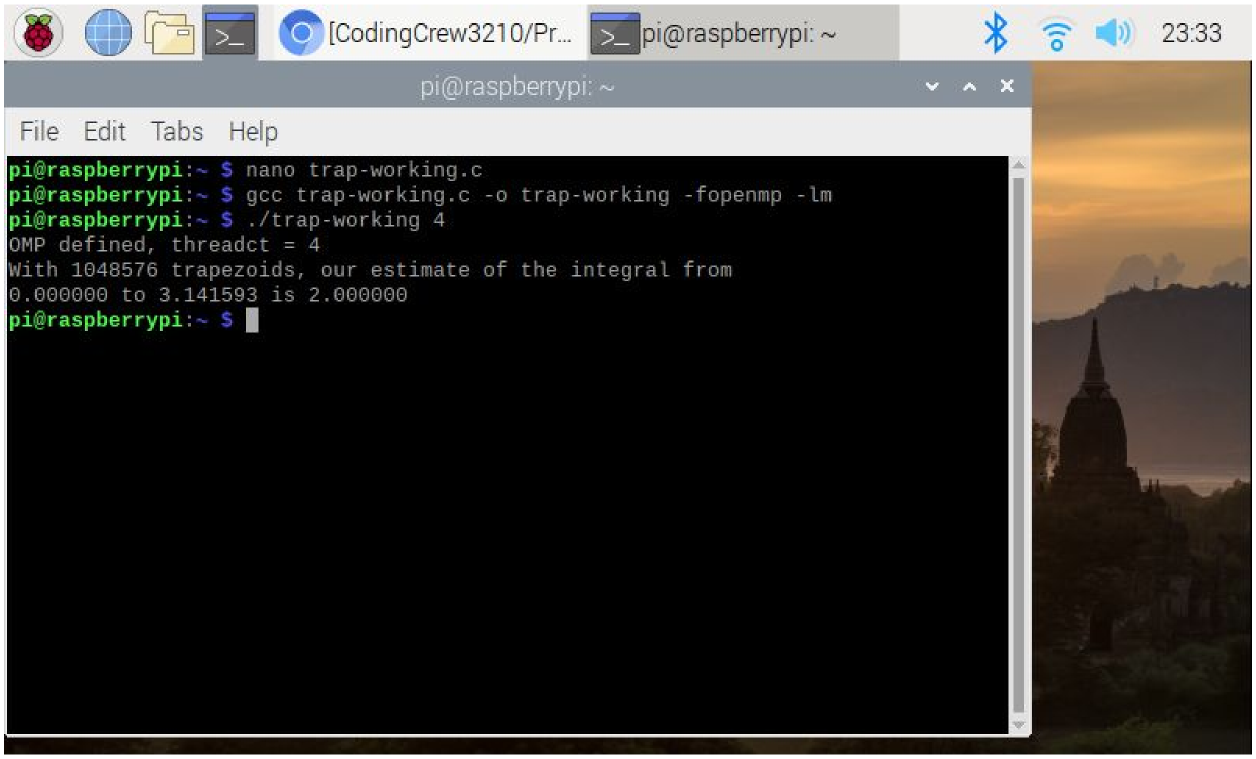
```
  GNU nano 3.2                    trap-working.c

#include <math.h>
#include <stdio.h> // printf()
#include <stdlib.h> // atoi()
#include <omp.h> // OpenMP
/*Demo program for OpenMP: computes trapezoidal approximation to an integral*/
const double pi = 3.141592653589793238462643383079;
int main(int argc, char** argv){
/*Variables*/
double a = 0.0, b = pi; /*limits of integration*/;
int n = 1048576; /*number of subdivisions = 2^20*/
double h = (b - a) / n; /*width of subdivision*/
double integral; /*accumulates answer*/
int threadcnt = 1;
double f(double x);
/*parse command-line arg for number of threads*/
if(argc > 1){
threadcnt = atoi(argv[1]);
}
                              [ Read 39 lines ]
^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
```



```
  GNU nano 3.2                    trap-working.c

#ifdef _OPENMP
omp_set_num_threads(threadcnt);
printf("OMP defined, threadct = %d\n",threadcnt);
#else
printf("OMP not defined");
#endif
integral = (f(a) + f(b))/2.0;
int i;
#pragma omp parallel for private(i) shared (a, n, h) reduction(+: integral)
for(i = 1; i < n; i++){
integral += f(a+i*h);
}
integral = integral * h;
printf("With %d trapezoids, our estimate of the integral from \n",n);
printf("%f to %f is %f\n",a,b,integral);
}
double f(double x){
return sin(x);

^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
```

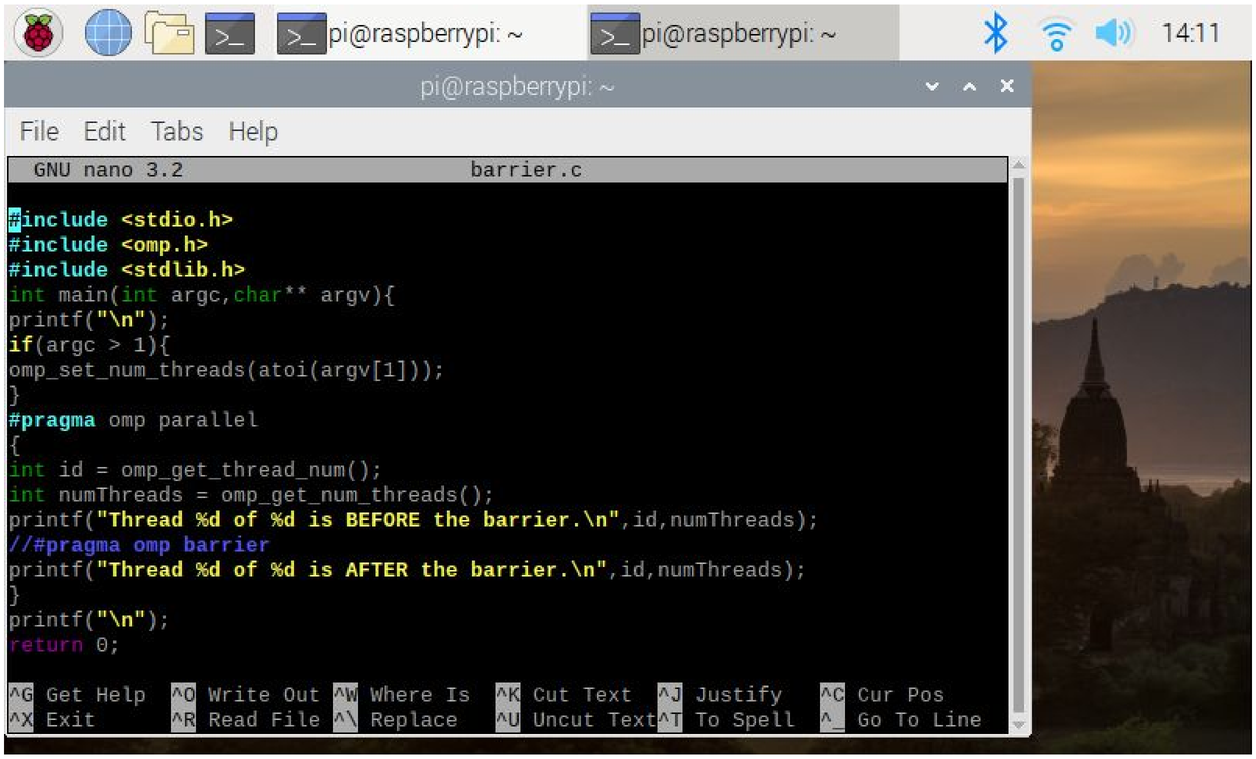


```
  GNU nano 3.2                        trap-working.c

for(i = 1; i < n; i++){
integral += f(a+i*h);
}
integral = integral * h;
printf("With %d trapezoids, our estimate of the integral from \n",n);
printf("%f to %f is %f\n",a,b,integral);
}
double f(double x){
return sin(x);
}

^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
```

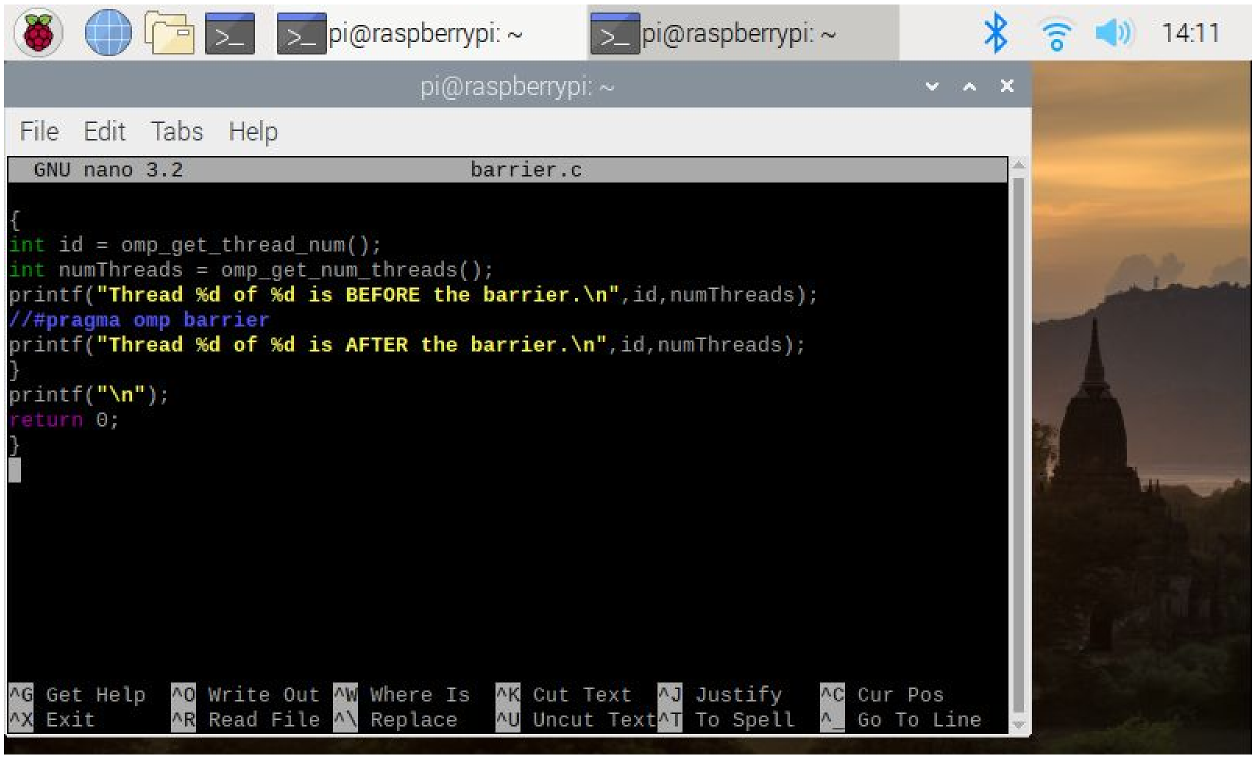


```
pi@raspberrypi:~ $ nano trap-working.c
pi@raspberrypi:~ $ gcc trap-working.c -o trap-working -fopenmp -lm
pi@raspberrypi:~ $ ./trap-working 4
OMP defined, threadct = 4
With 1048576 trapezoids, our estimate of the integral from
0.000000 to 3.141593 is 2.000000
pi@raspberrypi:~ $ 
```

For trap-working, I noticed a main difference between it and trap-not working and that was because of line 37

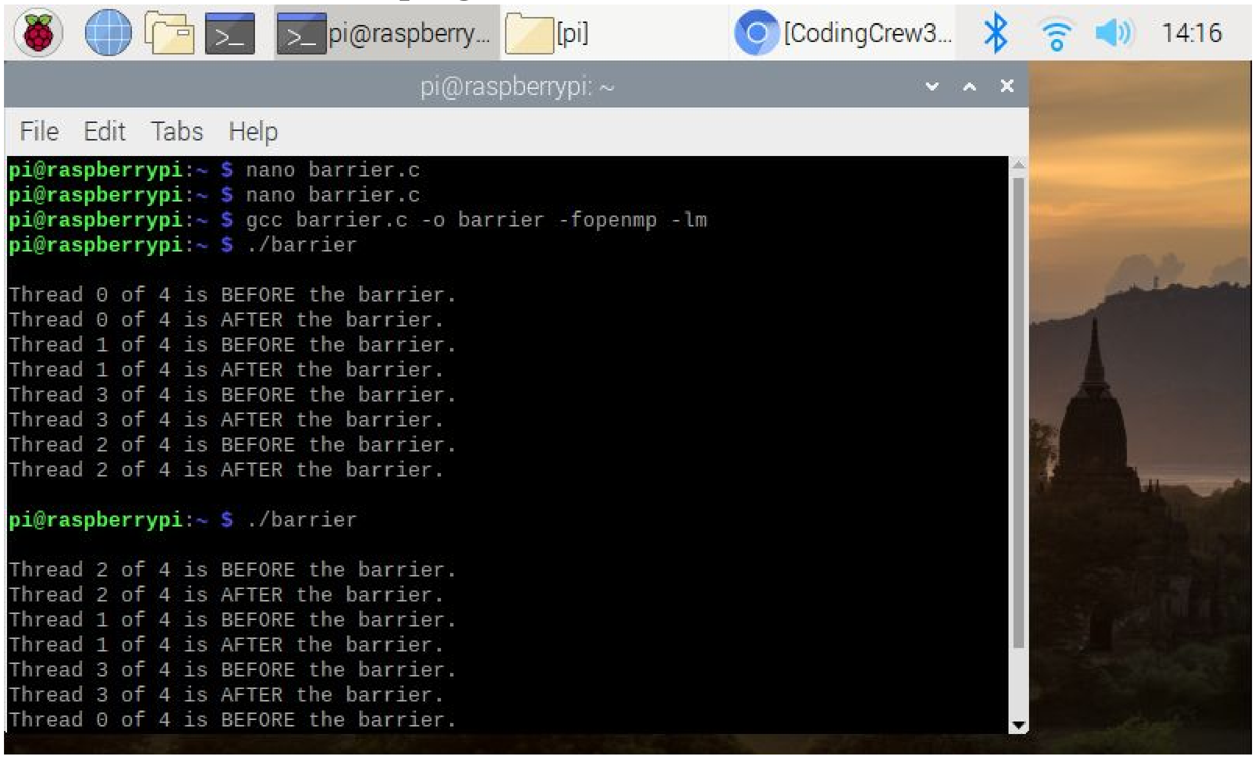


```
GNU nano 3.2                          barrier.c

#include <stdio.h>
#include <omp.h>
#include <stdlib.h>
int main(int argc,char** argv){
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("Thread %d of %d is BEFORE the barrier.\n",id,numThreads);
//#pragma omp barrier
printf("Thread %d of %d is AFTER the barrier.\n",id,numThreads);
}
printf("\n");
return 0;

^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
```

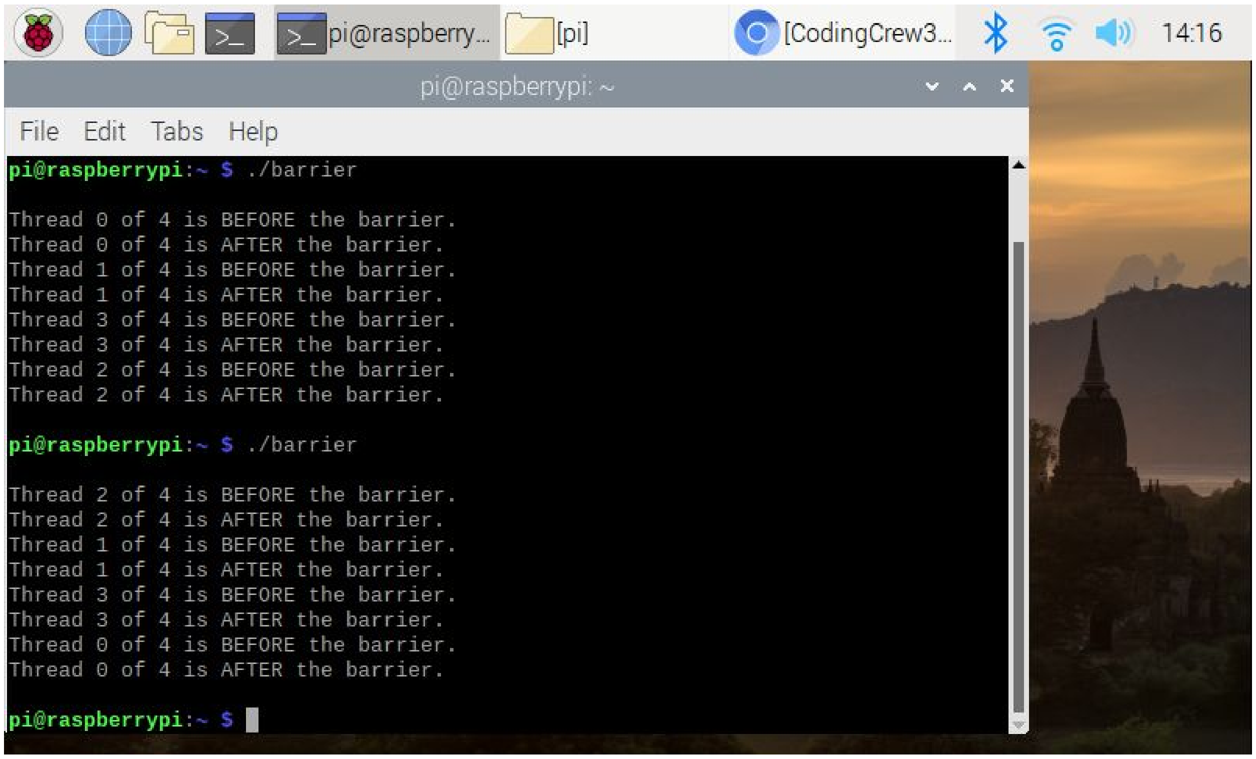


```
GNU nano 3.2                          barrier.c

{
int id = omp_get_thread_num();
int numThreads = omp_get_num_threads();
printf("Thread %d of %d is BEFORE the barrier.\n",id,numThreads);
//#pragma omp barrier
printf("Thread %d of %d is AFTER the barrier.\n",id,numThreads);
}
printf("\n");
return 0;
}

^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
```

Results with commented pragma on line 31:

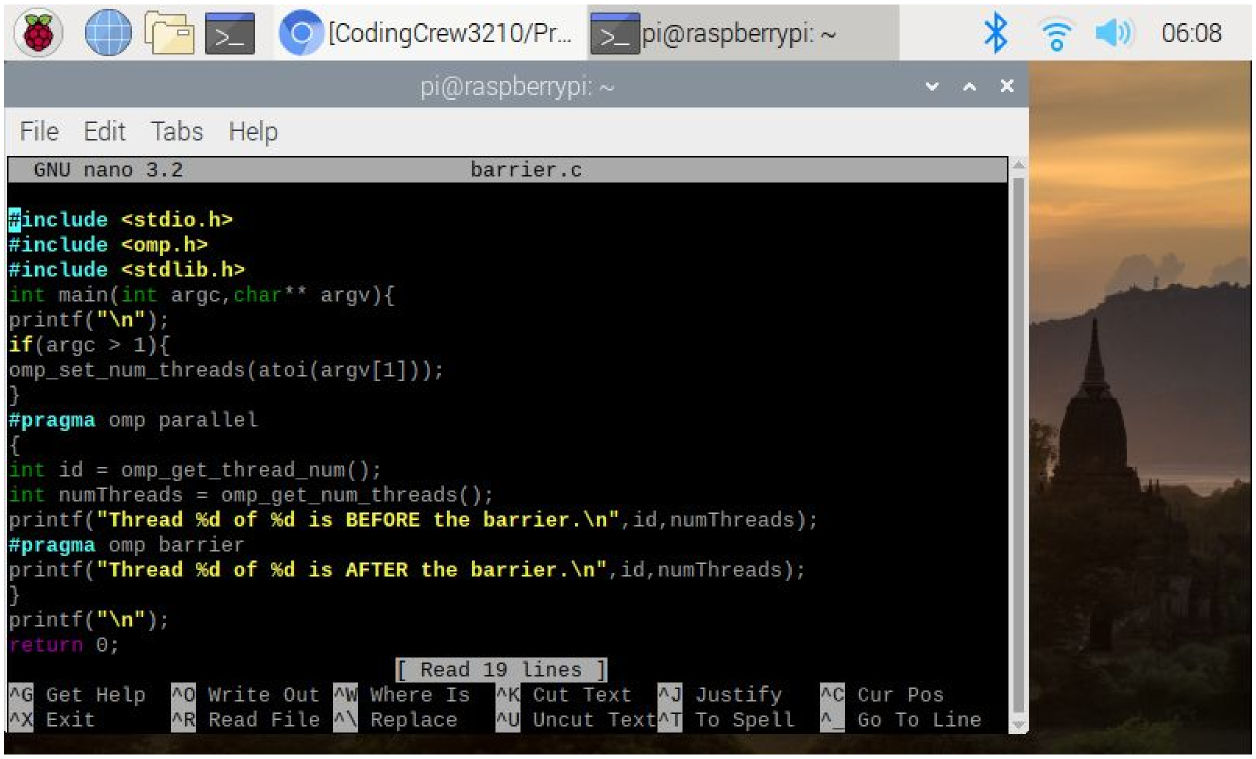


```
pi@raspberrypi:~ $ nano barrier.c
pi@raspberrypi:~ $ nano barrier.c
pi@raspberrypi:~ $ gcc barrier.c -o barrier -fopenmp -lm
pi@raspberrypi:~ $ ./barrier

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

pi@raspberrypi:~ $ ./barrier

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



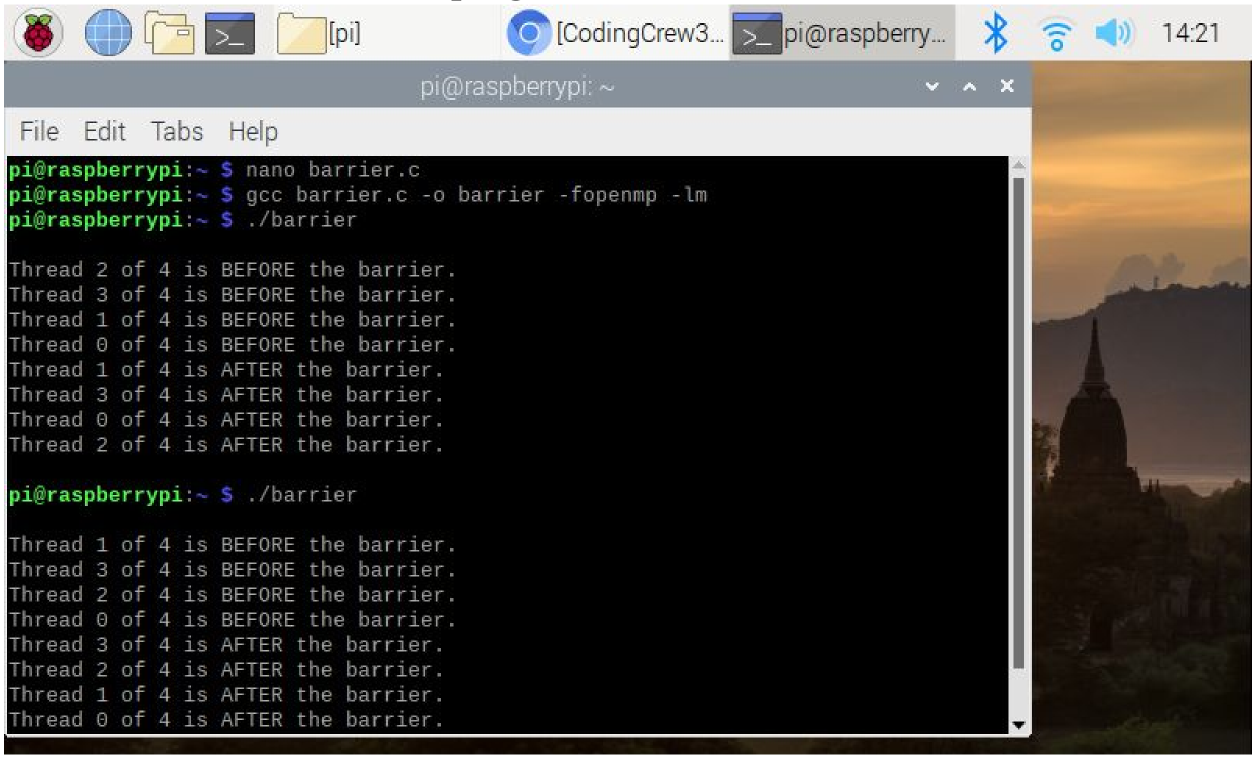
```
pi@raspberrypi:~ $ ./barrier

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

pi@raspberrypi:~ $ ./barrier

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

pi@raspberrypi:~ $
```

For barrier with the commented pragma,



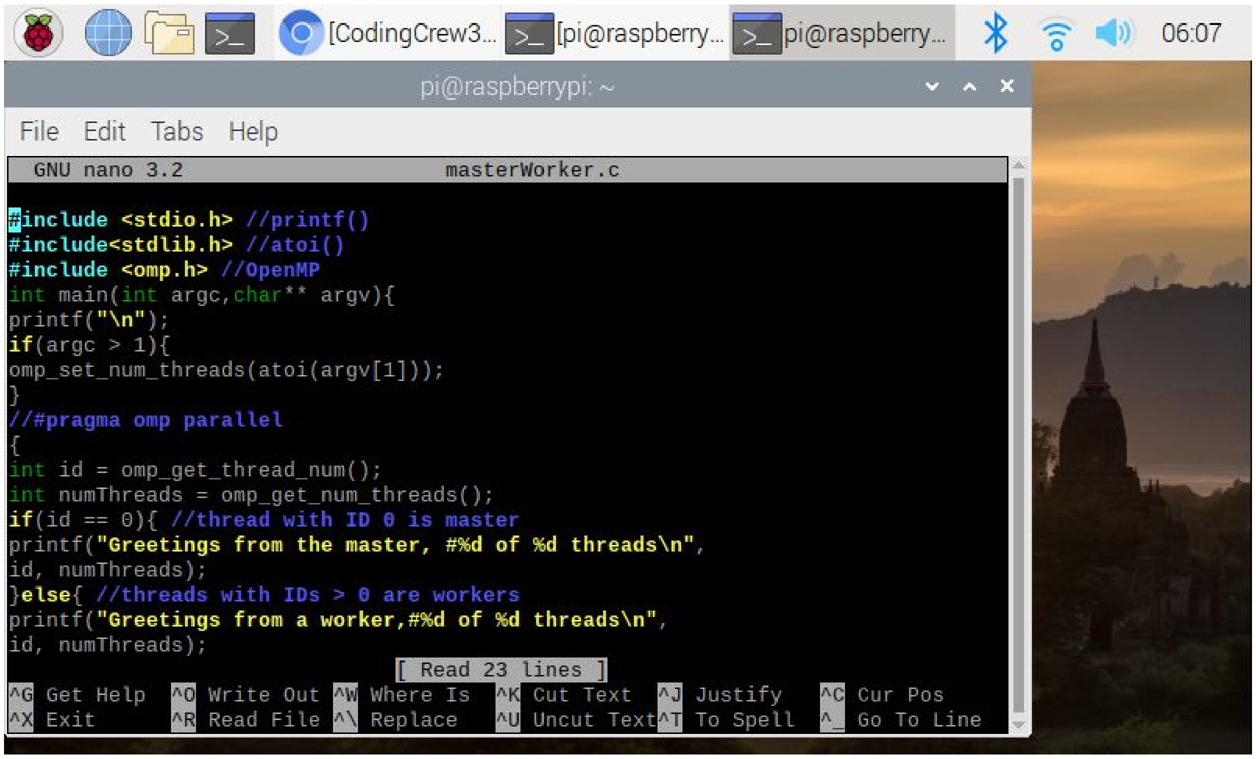
```
 GNU nano 3.2                          barrier.c

#include <stdio.h>
#include <omp.h>
#include <stdlib.h>
int main(int argc,char** argv){
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("Thread %d of %d is BEFORE the barrier.\n",id,numThreads);
#pragma omp barrier
printf("Thread %d of %d is AFTER the barrier.\n",id,numThreads);
}
printf("\n");
return 0;
                               [ Read 19 lines ]
^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
```

Results without commented pragma on line 31:

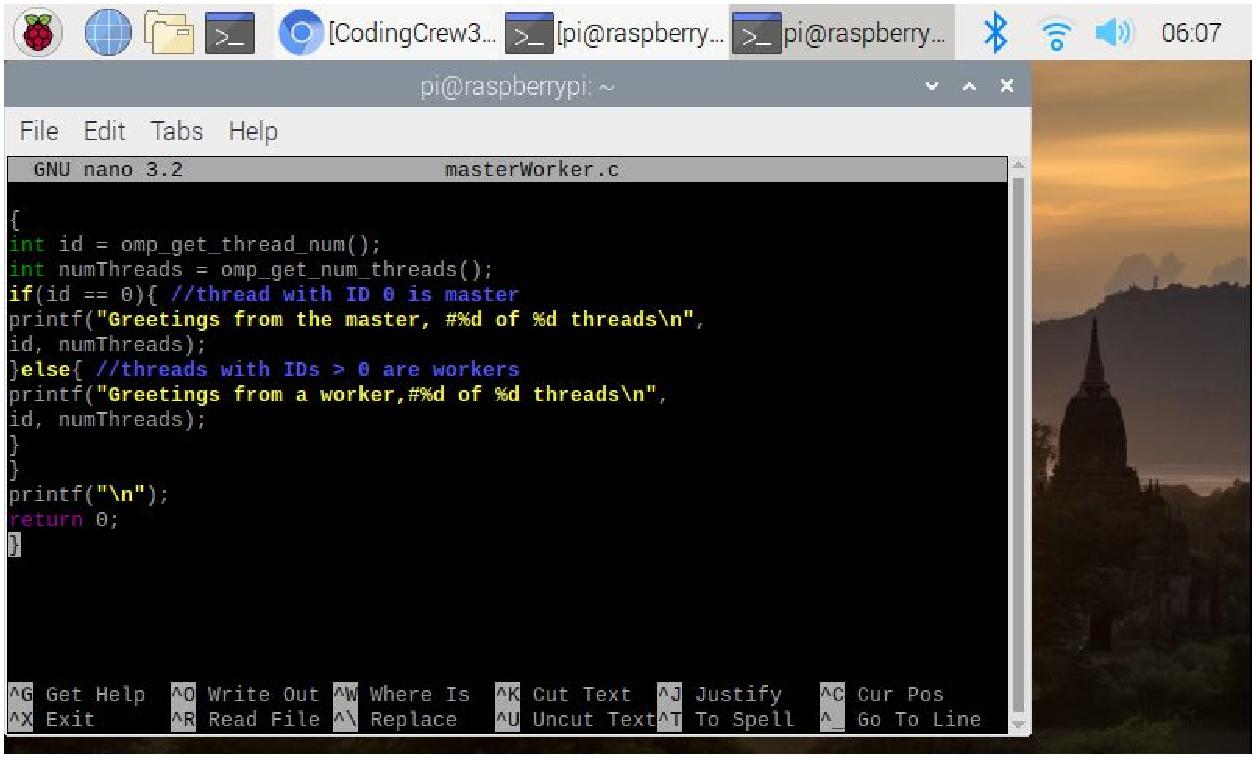


```
pi@raspberrypi:~ $ nano barrier.c
pi@raspberrypi:~ $ gcc barrier.c -o barrier -fopenmp -lm
pi@raspberrypi:~ $ ./barrier

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

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

For barrier without the commented pragma,

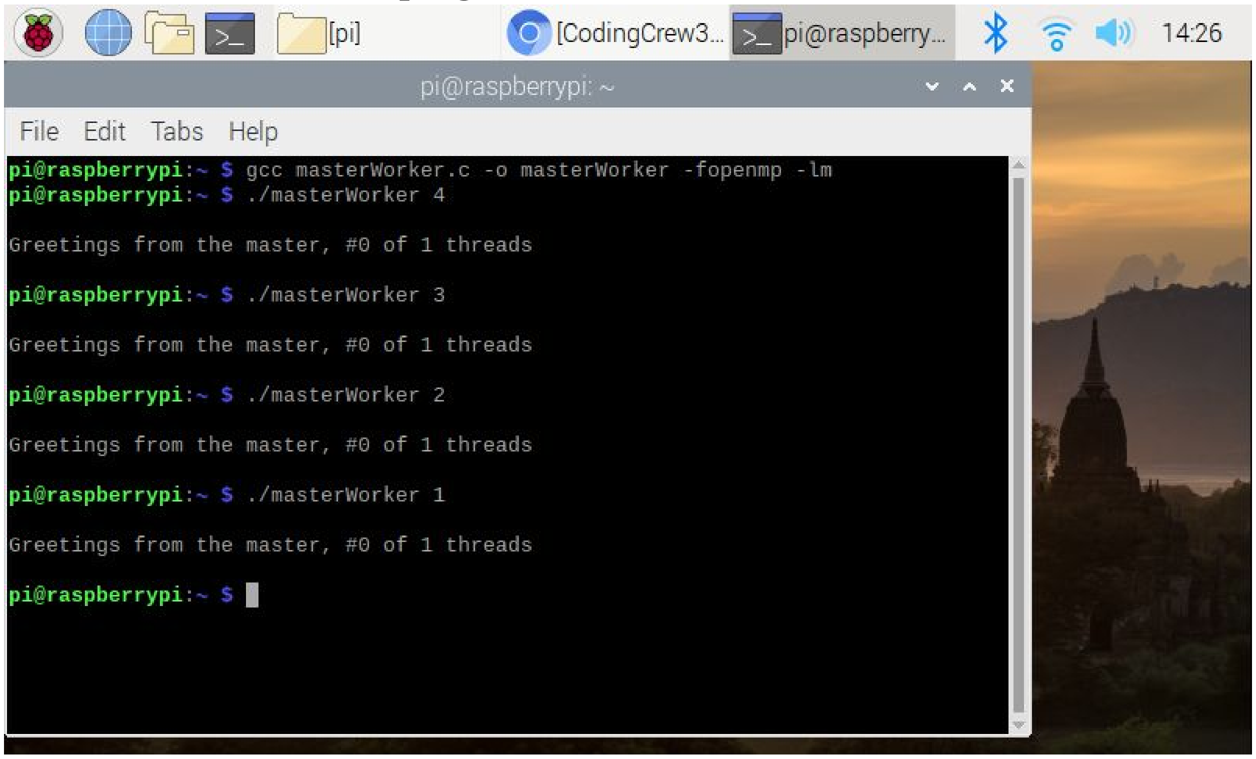


```
GNU nano 3.2                    masterWorker.c

#include <stdio.h> //printf()
#include<stdlib.h> //atoi()
#include <omp.h> //OpenMP
int main(int argc,char** argv){
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();
if(id == 0){ //thread with ID 0 is master
printf("Greetings from the master, #%d of %d threads\n",
id, numThreads);
}else{ //threads with IDs > 0 are workers
printf("Greetings from a worker,#%d of %d threads\n",
id, numThreads);
                               [ Read 23 lines ]
^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
```



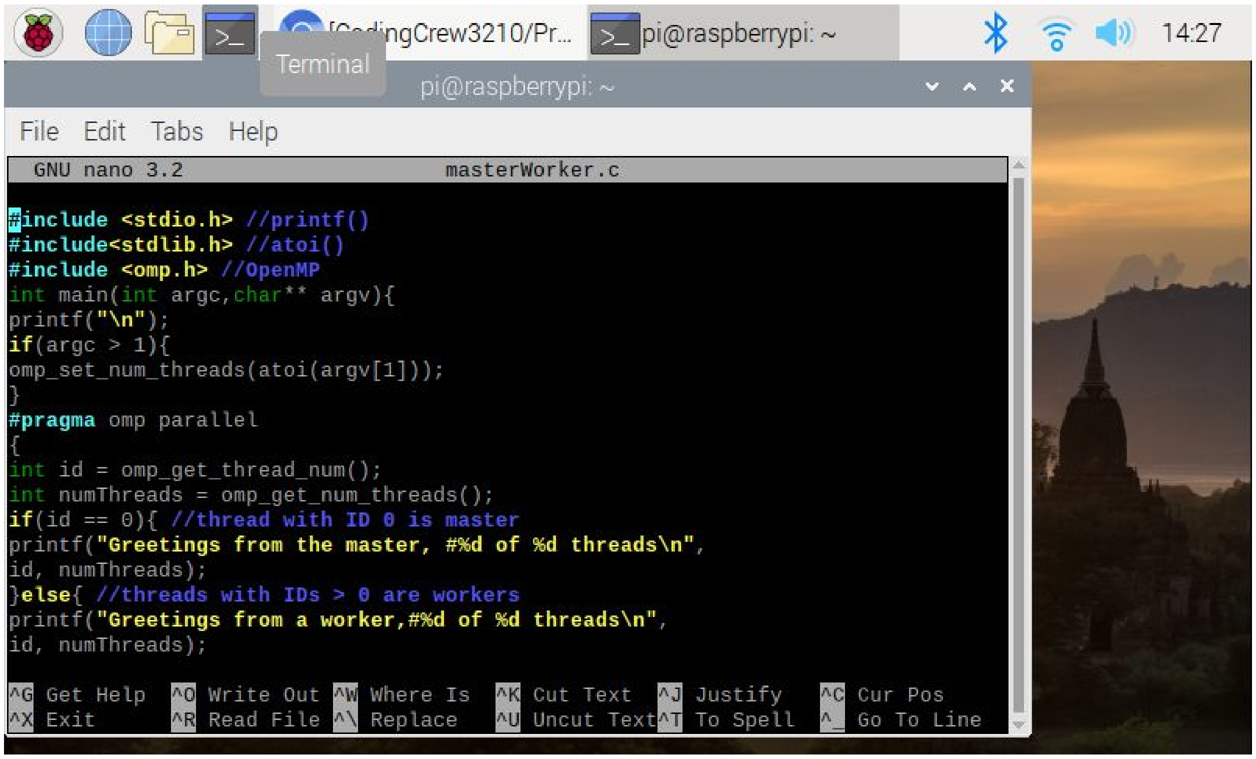
```
GNU nano 3.2                    masterWorker.c

{
int id = omp_get_thread_num();
int numThreads = omp_get_num_threads();
if(id == 0){ //thread with ID 0 is master
printf("Greetings from the master, #%d of %d threads\n",
id, numThreads);
}else{ //threads with IDs > 0 are workers
printf("Greetings from a worker,#%d of %d threads\n",
id, numThreads);
}
}
printf("\n");
return 0;
}

^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
```

Results with commented pragma on line 24:

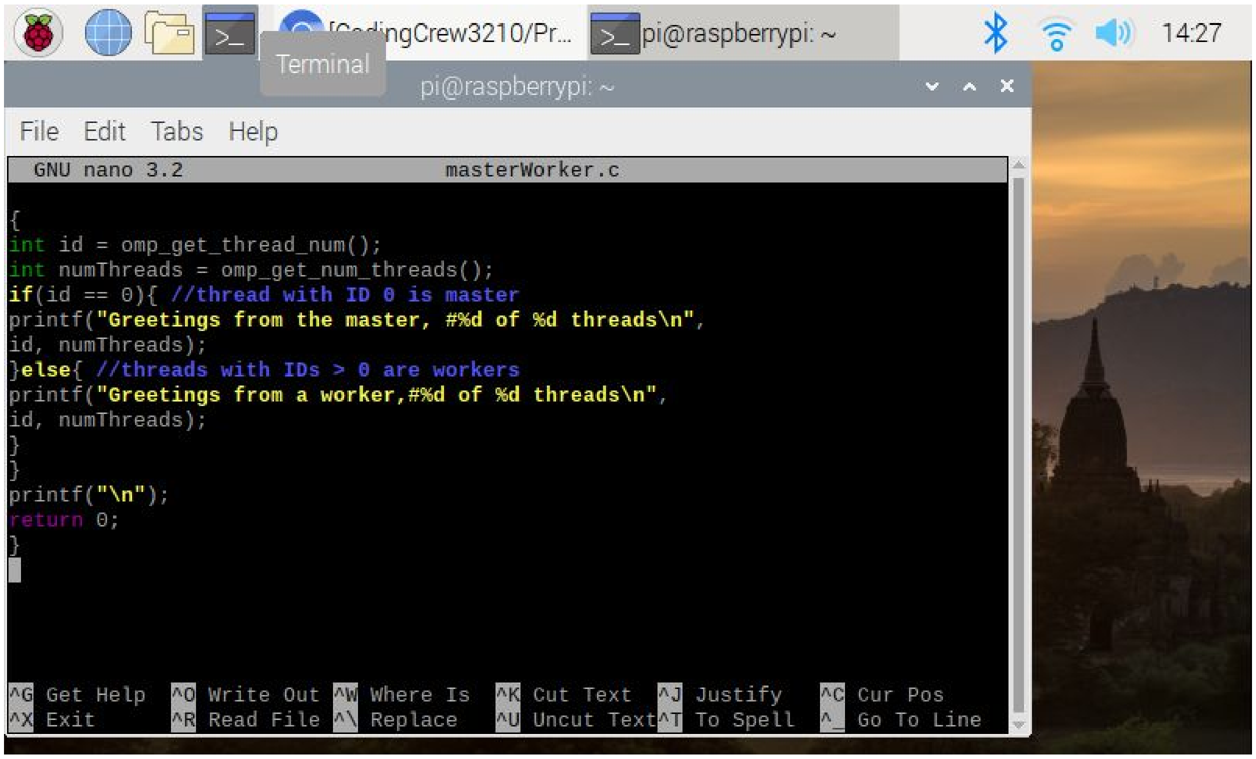


```
pi@raspberrypi:~ $ gcc masterWorker.c -o masterWorker -fopenmp -lm
pi@raspberrypi:~ $ ./masterWorker 4

Greetings from the master, #0 of 1 threads

pi@raspberrypi:~ $ ./masterWorker 3

Greetings from the master, #0 of 1 threads

pi@raspberrypi:~ $ ./masterWorker 2

Greetings from the master, #0 of 1 threads

pi@raspberrypi:~ $ ./masterWorker 1

Greetings from the master, #0 of 1 threads

pi@raspberrypi:~ $ 
```

For masterWorker with the commented pragma,

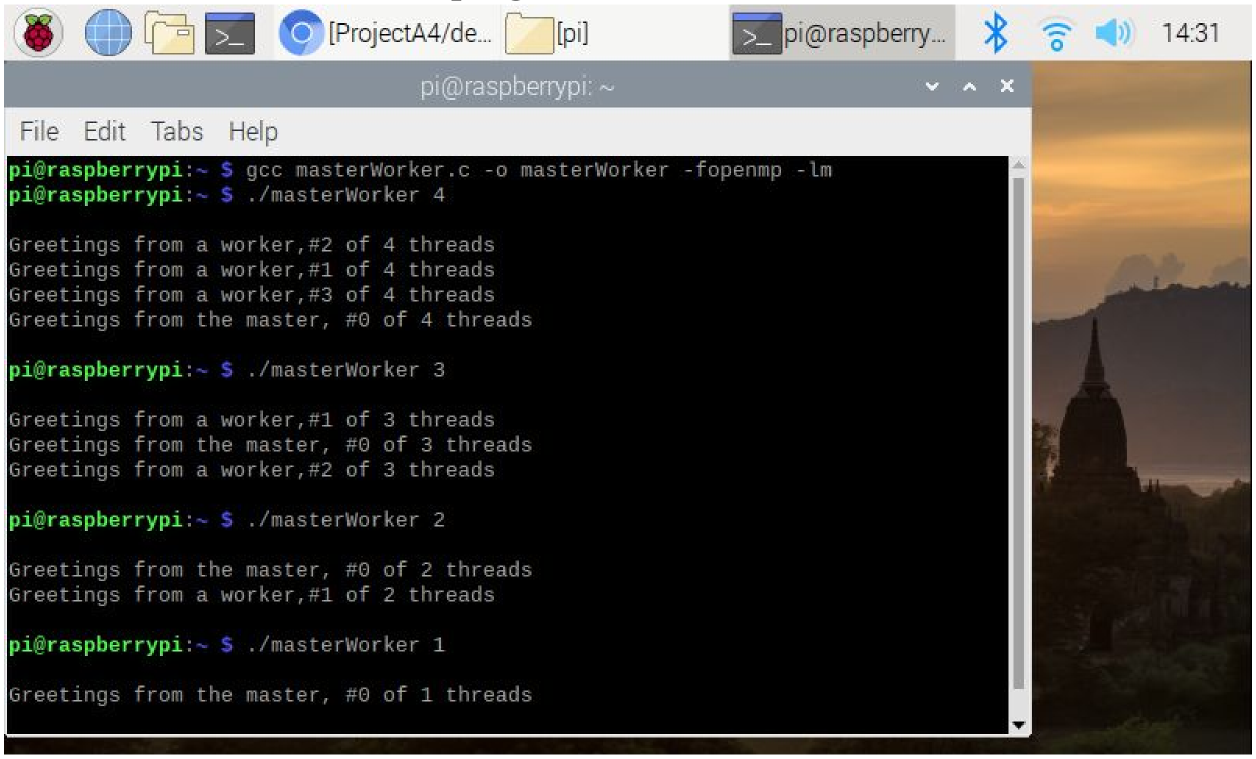


```
  GNU nano 3.2                          masterWorker.c

#include <stdio.h> //printf()
#include<stdlib.h> //atoi()
#include <omp.h> //OpenMP
int main(int argc,char** argv){
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();
if(id == 0){ //thread with ID 0 is master
printf("Greetings from the master, #%d of %d threads\n",
id, numThreads);
}else{ //threads with IDs > 0 are workers
printf("Greetings from a worker,#%d of %d threads\n",
id, numThreads);

^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
```



```
 GNU nano 3.2                    masterWorker.c

{
int id = omp_get_thread_num();
int numThreads = omp_get_num_threads();
if(id == 0){ //thread with ID 0 is master
printf("Greetings from the master, #%d of %d threads\n",
id, numThreads);
}else{ //threads with IDs > 0 are workers
printf("Greetings from a worker,#%d of %d threads\n",
id, numThreads);
}
}
printf("\n");
return 0;
}

^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
```

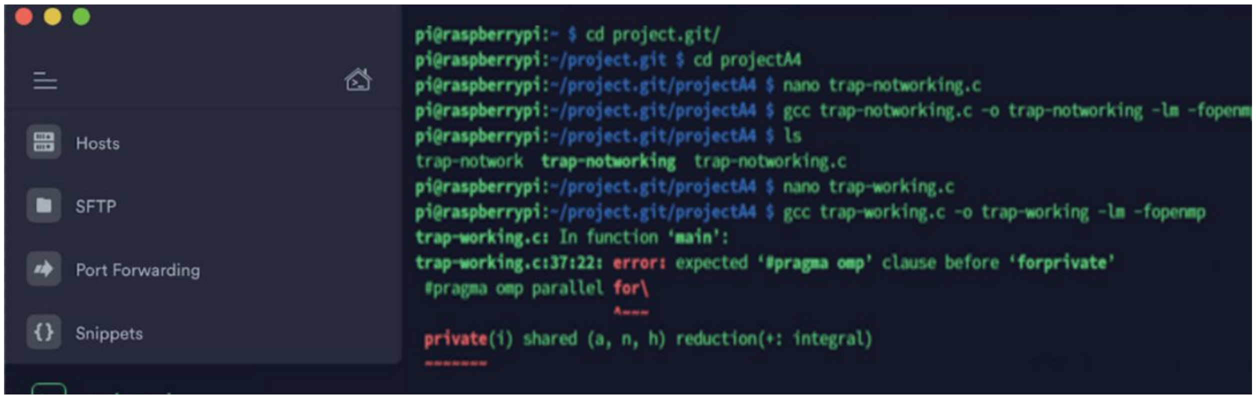
Results without commented pragma on line 24:



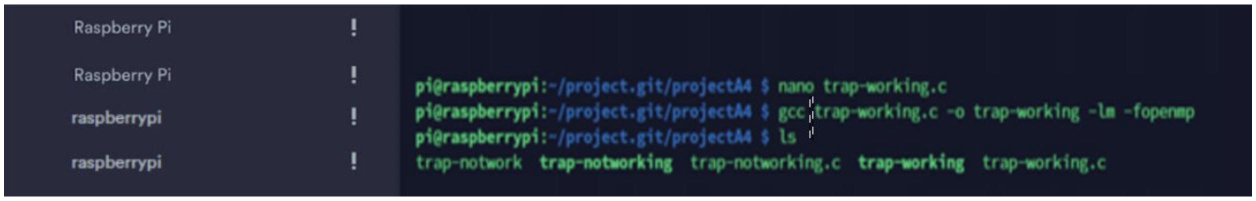
```
pi@raspberrypi:~ $ gcc masterWorker.c -o masterWorker -fopenmp -lm
pi@raspberrypi:~ $ ./masterWorker 4

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

pi@raspberrypi:~ $ ./masterWorker 3

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

pi@raspberrypi:~ $ ./masterWorker 2

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

pi@raspberrypi:~ $ ./masterWorker 1

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

For masterWorker without the commented pragma,

Mohammad Munsur (Rohan):



```
pi@raspberrypi:~ $ cd project.git/
pi@raspberrypi:~/project.git $ cd projectA4
pi@raspberrypi:~/project.git/projectA4 $ nano trap-notworking.c
pi@raspberrypi:~/project.git/projectA4 $ gcc trap-notworking.c -o trap-notworking -lm -fopenm
pi@raspberrypi:~/project.git/projectA4 $ ls
trap-notwork  trap-notworking  trap-notworking.c
pi@raspberrypi:~/project.git/projectA4 $ nano trap-working.c
pi@raspberrypi:~/project.git/projectA4 $ gcc trap-working.c -o trap-working -lm -fopenmp
trap-working.c: In function 'main':
trap-working.c:37:22: error: expected '#pragma omp' clause before 'forprivate'
 #pragma omp parallel for\
                      ^~~~
 private(i) shared (a, n, h) reduction(+: integral)
 ~~~~~~~
```

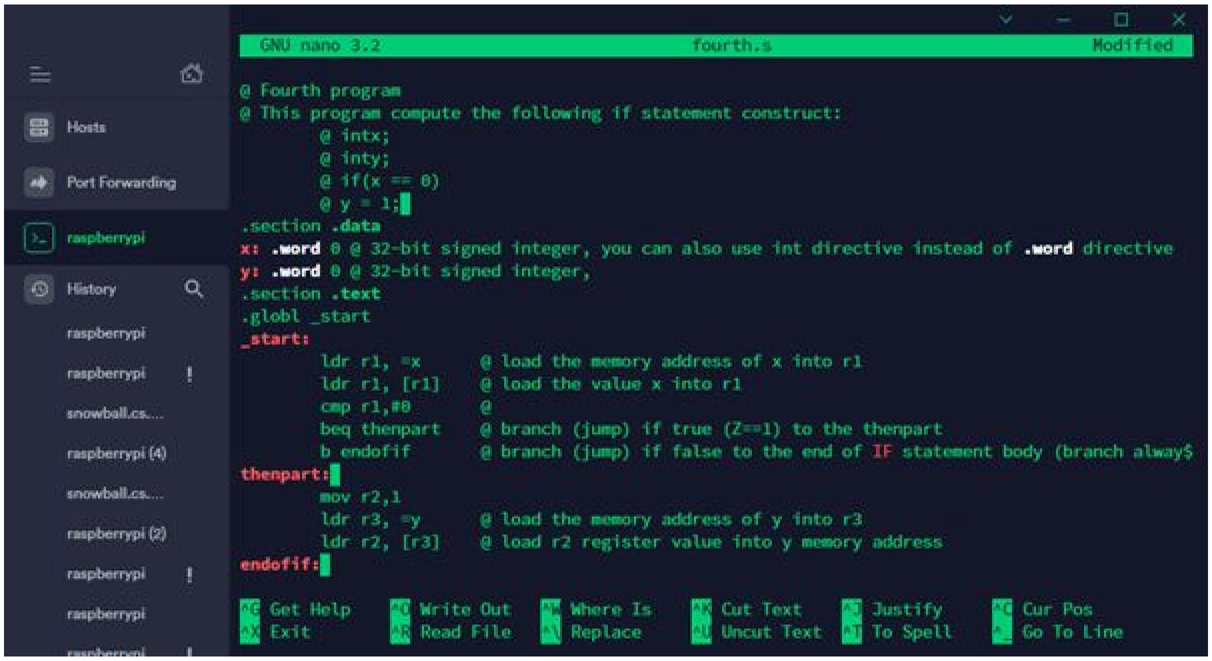


```
pi@raspberrypi:~/project.git/projectA4 $ nano trap-working.c
pi@raspberrypi:~/project.git/projectA4 $ gcc trap-working.c -o trap-working -lm -fopenmp
pi@raspberrypi:~/project.git/projectA4 $ ls
trap-notwork  trap-notworking  trap-notworking.c  trap-working  trap-working.c
```

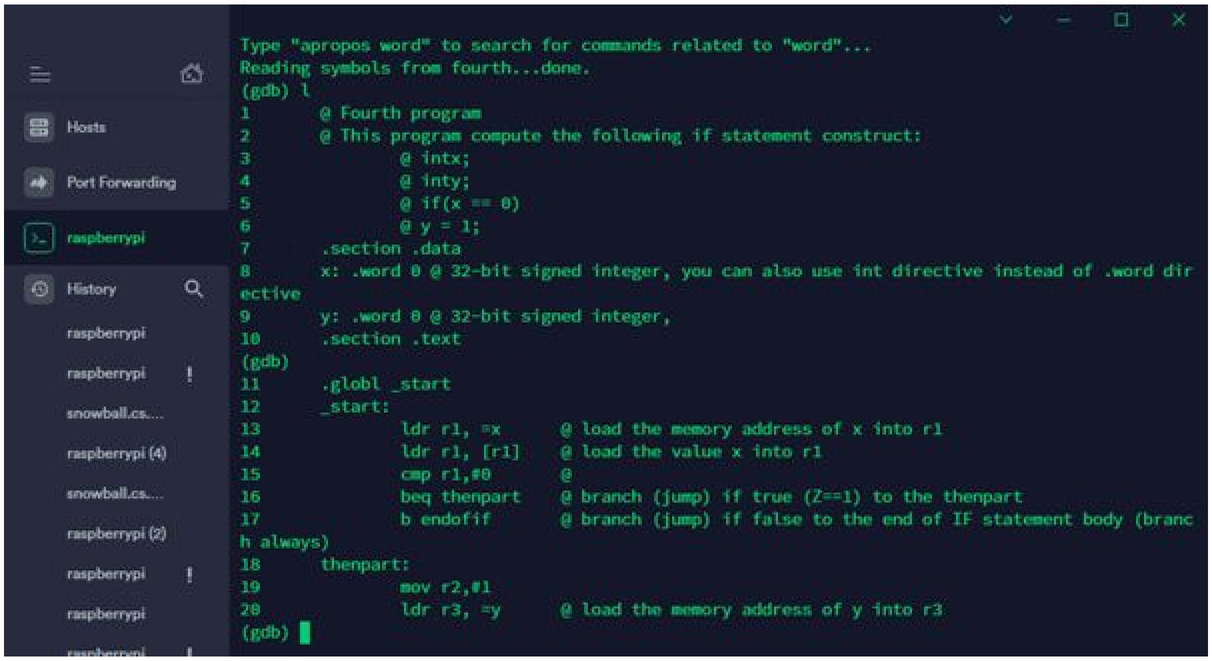
ARM Assembly Programming

Ping He:

Part One:

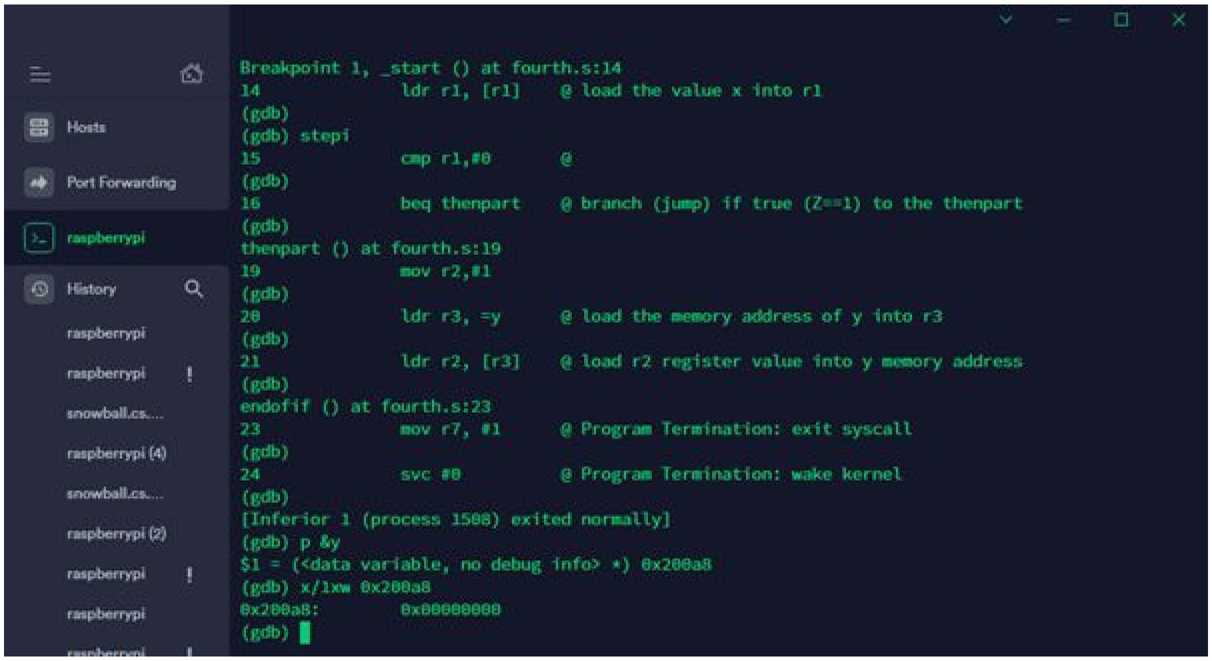


```
GNU nano 3.2                          fourth.s                          Modified

@ Fourth program
@ This program compute the following if statement construct:
        @ intx;
        @ inty;
        @ if(x == 0)
        @ y = 1;
.section .data
x: .word 0 @ 32-bit signed integer, you can also use int directive instead of .word directive
y: .word 0 @ 32-bit signed integer,
.section .text
.globl _start
_start:
        ldr r1, =x      @ load the memory address of x into r1
        ldr r1, [r1]    @ load the value x into r1
        cmp r1,#0       @
        beq thenpart    @ branch (jump) if true (Z==1) to the thenpart
        b endofif       @ branch (jump) if false to the end of IF statement body (branch alway$
thenpart:
        mov r2,1
        ldr r3, =y      @ load the memory address of y into r3
        ldr r2, [r3]    @ load r2 register value into y memory address
endofif:

^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
```



```
Type "apropos word" to search for commands related to "word"...
Reading symbols from fourth...done.
(gdb) l
1       @ Fourth program
2       @ This program compute the following if statement construct:
3               @ intx;
4               @ inty;
5               @ if(x == 0)
6               @ y = 1;
7       .section .data
8       x: .word 0 @ 32-bit signed integer, you can also use int directive instead of .word dir
ective
9       y: .word 0 @ 32-bit signed integer,
10      .section .text
(gdb)
11      .globl _start
12      _start:
13              ldr r1, =x      @ load the memory address of x into r1
14              ldr r1, [r1]    @ load the value x into r1
15              cmp r1,#0       @
16              beq thenpart    @ branch (jump) if true (Z==1) to the thenpart
17              b endofif       @ branch (jump) if false to the end of IF statement body (branc
h always)
18      thenpart:
19              mov r2,#1
20              ldr r3, =y      @ load the memory address of y into r3
(gdb)
```

Set the breakpoint at line 14:

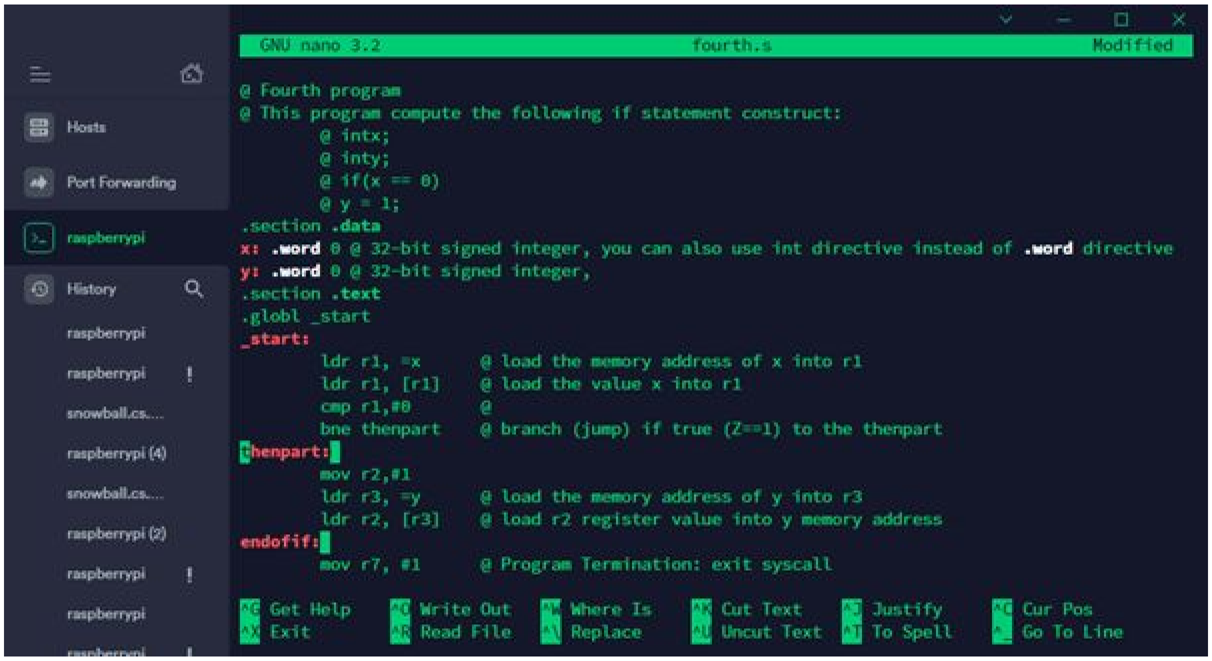


```
Breakpoint 1, _start () at fourth.s:14
14              ldr r1, [r1]    @ load the value x into r1
(gdb)
(gdb) stepi
15              cmp r1,#0       @
(gdb)
16              beq thenpart    @ branch (jump) if true (Z==1) to the thenpart
(gdb)
thenpart () at fourth.s:19
19              mov r2,#1
(gdb)
20              ldr r3, =y      @ load the memory address of y into r3
(gdb)
21              ldr r2, [r3]    @ load r2 register value into y memory address
(gdb)
endofif () at fourth.s:23
23              mov r7, #1      @ Program Termination: exit syscall
(gdb)
24              svc #0          @ Program Termination: wake kernel
(gdb)
[Inferior 1 (process 1508) exited normally]
(gdb) p &y
$1 = (<data variable, no debug info> *) 0x200a8
(gdb) x/1xw 0x200a8
0x200a8:        0x00000000
(gdb)
```

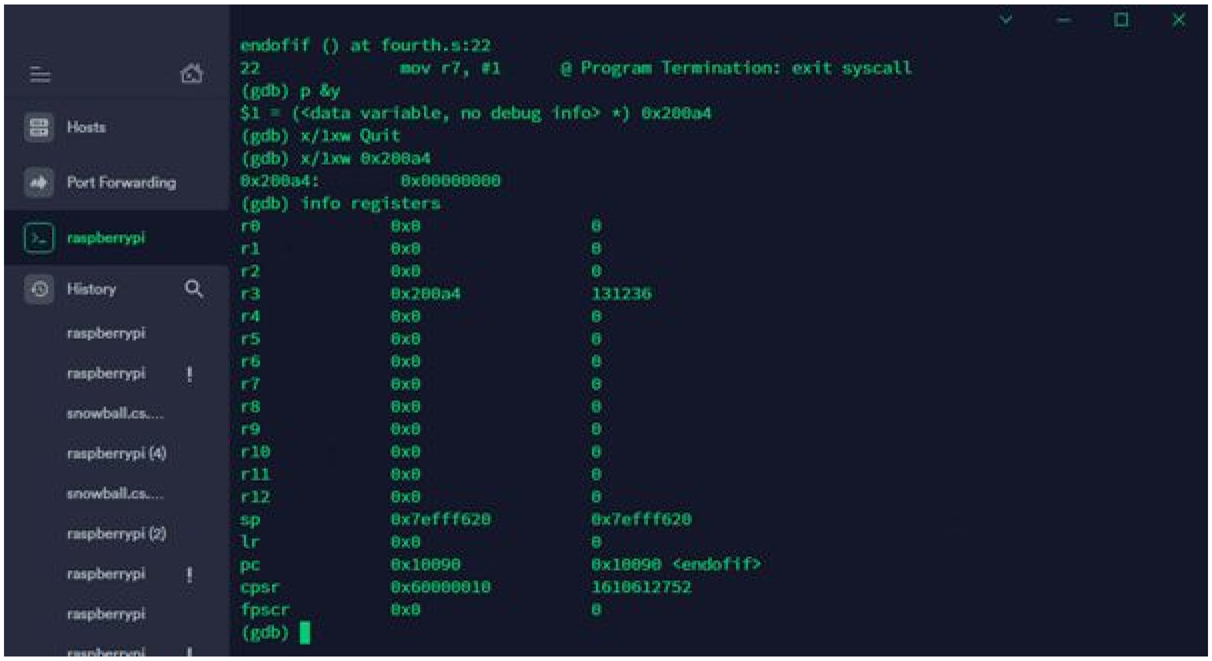
The value that is stored in the memory of y is 0, because we didn’t change the value of y; the value of the Z flag is 1, which is right. (cpsr=0x60000010)

Part Two:

Replace beq with bnq and remove b instruction:



```
GNU nano 3.2                          fourth.s                          Modified

@ Fourth program
@ This program compute the following if statement construct:
        @ intx;
        @ inty;
        @ if(x == 0)
        @ y = 1;
.section .data
x: .word 0 @ 32-bit signed integer, you can also use int directive instead of .word directive
y: .word 0 @ 32-bit signed integer,
.section .text
.globl _start
_start:
        ldr r1, =x      @ load the memory address of x into r1
        ldr r1, [r1]    @ load the value x into r1
        cmp r1,#0       @
        bne thenpart    @ branch (jump) if true (Z==1) to the thenpart
thenpart:
        mov r2,#1
        ldr r3, =y      @ load the memory address of y into r3
        ldr r2, [r3]    @ load r2 register value into y memory address
endofif:
        mov r7, #1      @ Program Termination: exit syscall
```



```
endofif () at fourth.s:22
22              mov r7, #1      @ Program Termination: exit syscall
(gdb) p &y
$1 = (<data variable, no debug info> *) 0x200a4
(gdb) x/1xw Quit
(gdb) x/1xw 0x200a4
0x200a4:        0x00000000
(gdb) info registers
r0             0x0                 0
r1             0x0                 0
r2             0x0                 0
r3             0x200a4             131236
r4             0x0                 0
r5             0x0                 0
r6             0x0                 0
r7             0x0                 0
r8             0x0                 0
r9             0x0                 0
r10            0x0                 0
r11            0x0                 0
r12            0x0                 0
sp             0x7efff620          0x7efff620
lr             0x0                 0
pc             0x10090             0x10090 <endofif>
cpsr           0x60000010          1610612752
fpscr          0x0                 0
(gdb)
```

The updated program is efficient because we replaced the part that contains back-to-back branches. The result of y is still 0 and the value of the Z flag is 1.

<u>Part Three:</u>

My code:

@ Control Structures

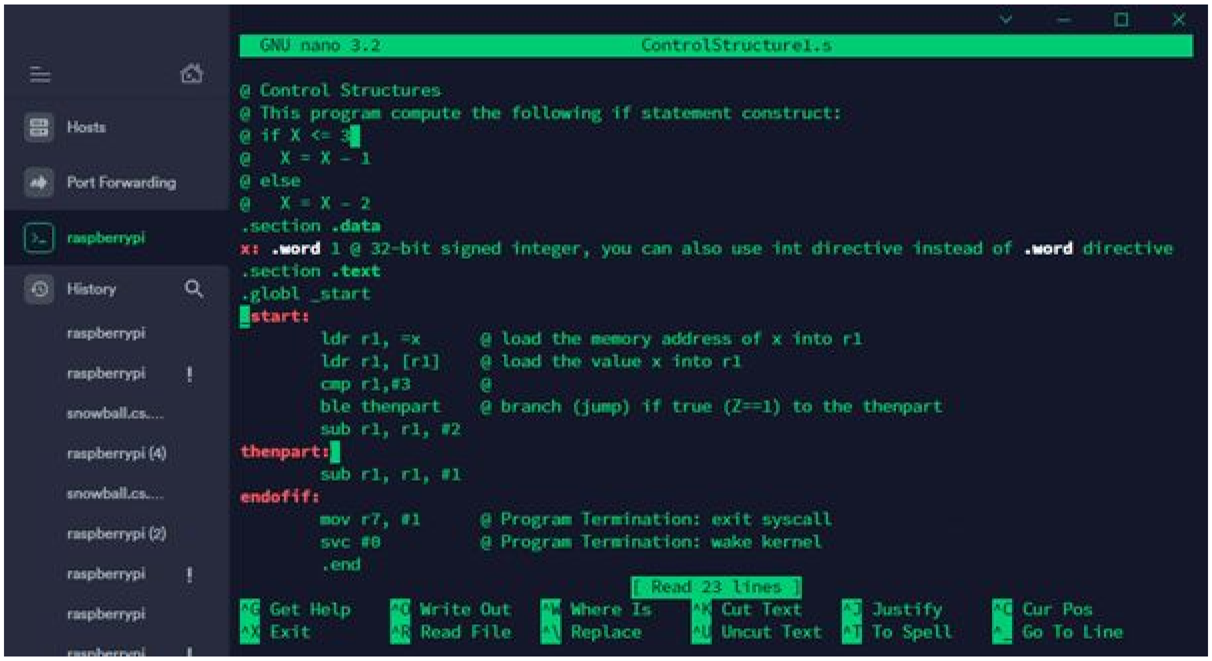
@ This program compute the following if statement construct:

@ if X <= 3

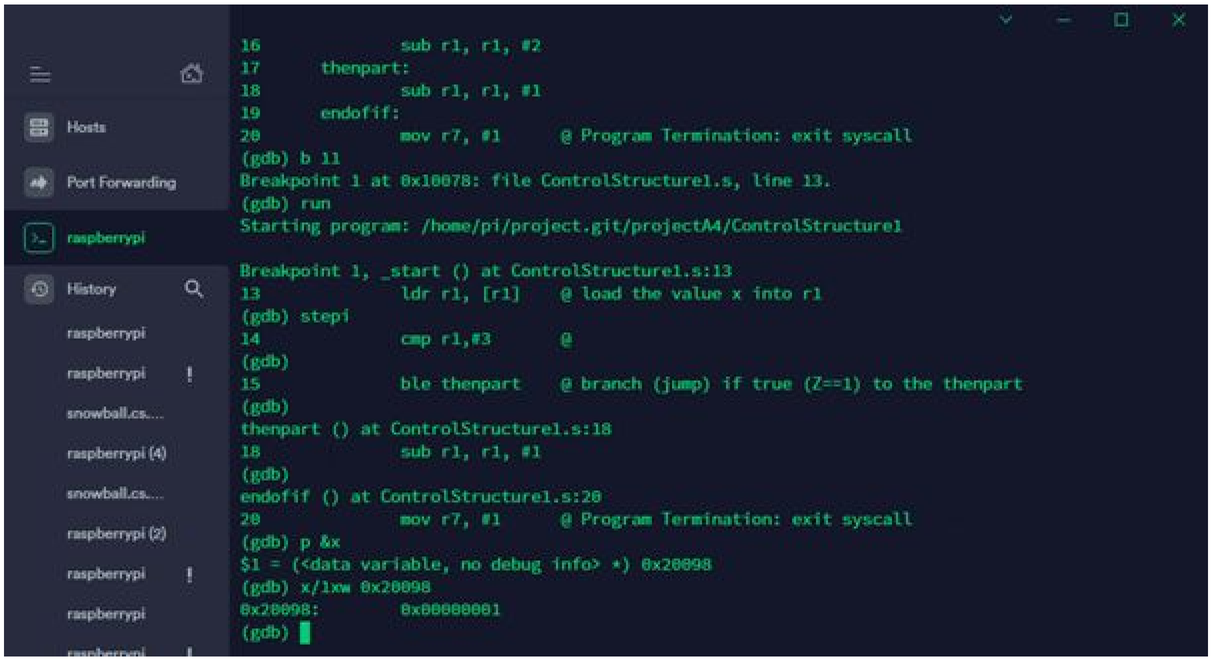
@   X = X – 1

@ else

```
@   X = X – 2
.section .data
x: .word 1 @ 32-bit signed integer, you can also use int directive instead of .word directive
.section .text
.globl _start
_start:
        ldr r1, =x       @ load the memory address of x into r1
        ldr r1, [r1]     @ load the value x into r1
        cmp r1,#3        @
        ble thenpart     @ branch (jump) if true (Z==1) to the thenpart
        sub r1, r1, #2
thenpart:
        sub r1, r1, #1
endofif:
        mov r7, #1       @ Program Termination: exit syscall
        svc #0           @ Program Termination: wake kernel
        .end
```

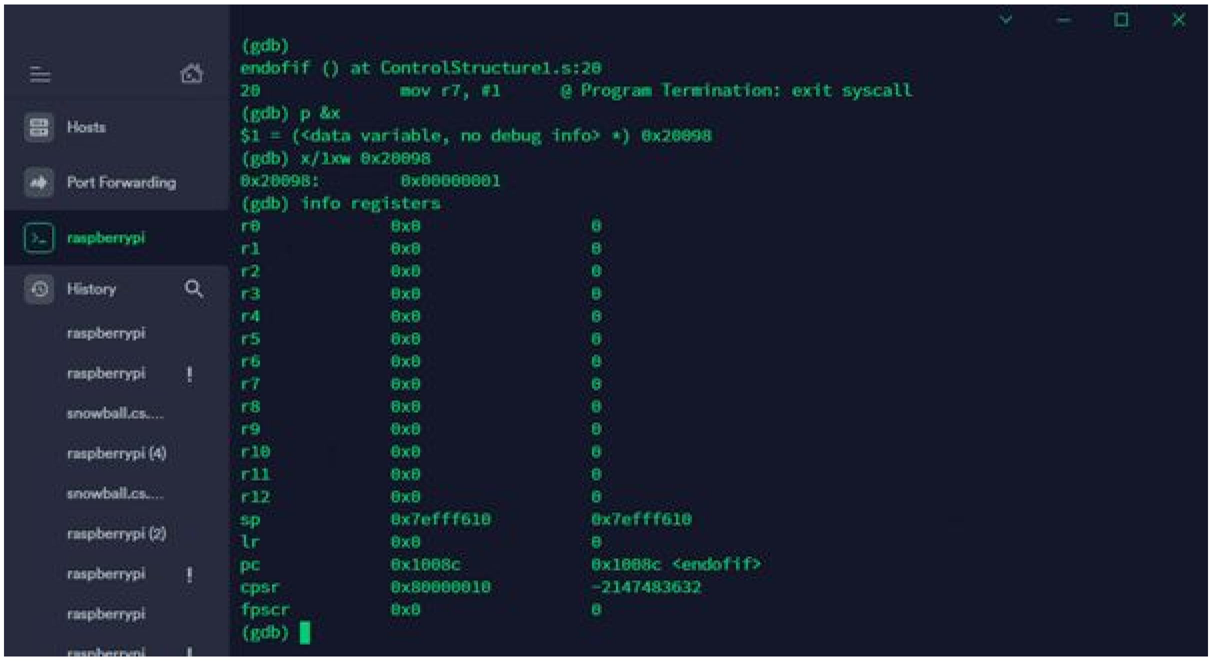


```
  GNU nano 3.2                        ControlStructure1.s

@ Control Structures
@ This program compute the following if statement construct:
@ if X <= 3
@   X = X - 1
@ else
@   X = X - 2
.section .data
x: .word 1 @ 32-bit signed integer, you can also use int directive instead of .word directive
.section .text
.globl _start
_start:
        ldr r1, =x      @ load the memory address of x into r1
        ldr r1, [r1]    @ load the value x into r1
        cmp r1,#3       @
        ble thenpart    @ branch (jump) if true (Z==1) to the thenpart
        sub r1, r1, #2
thenpart:
        sub r1, r1, #1
endofif:
        mov r7, #1      @ Program Termination: exit syscall
        svc #0          @ Program Termination: wake kernel
        .end
                                [ Read 23 lines ]
^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
```



```
16          sub r1, r1, #2
17      thenpart:
18          sub r1, r1, #1
19      endofif:
20          mov r7, #1      @ Program Termination: exit syscall
(gdb) b 11
Breakpoint 1 at 0x10078: file ControlStructure1.s, line 13.
(gdb) run
Starting program: /home/pi/project.git/projectA4/ControlStructure1

Breakpoint 1, _start () at ControlStructure1.s:13
13          ldr r1, [r1]    @ load the value x into r1
(gdb) stepi
14          cmp r1,#3       @
(gdb)
15          ble thenpart    @ branch (jump) if true (Z==1) to the thenpart
(gdb)
thenpart () at ControlStructure1.s:18
18          sub r1, r1, #1
(gdb)
endofif () at ControlStructure1.s:20
20          mov r7, #1      @ Program Termination: exit syscall
(gdb) p &x
$1 = (<data variable, no debug info> *) 0x20098
(gdb) x/1xw 0x20098
0x20098:        0x00000001
(gdb) 
```

The value of x is 1, so while execute, it will jump into the then part.



```
(gdb)
endofif () at ControlStructure1.s:20
20          mov r7, #1      @ Program Termination: exit syscall
(gdb) p &x
$1 = (<data variable, no debug info> *) 0x20098
(gdb) x/1xw 0x20098
0x20098:        0x00000001
(gdb) info registers
r0             0x0                 0
r1             0x0                 0
r2             0x0                 0
r3             0x0                 0
r4             0x0                 0
r5             0x0                 0
r6             0x0                 0
r7             0x0                 0
r8             0x0                 0
r9             0x0                 0
r10            0x0                 0
r11            0x0                 0
r12            0x0                 0
sp             0x7efff610          0x7efff610
lr             0x0                 0
pc             0x1008c             0x1008c <endofif>
cpsr           0x80000010          -2147483632
fpscr          0x0                 0
(gdb) 
```

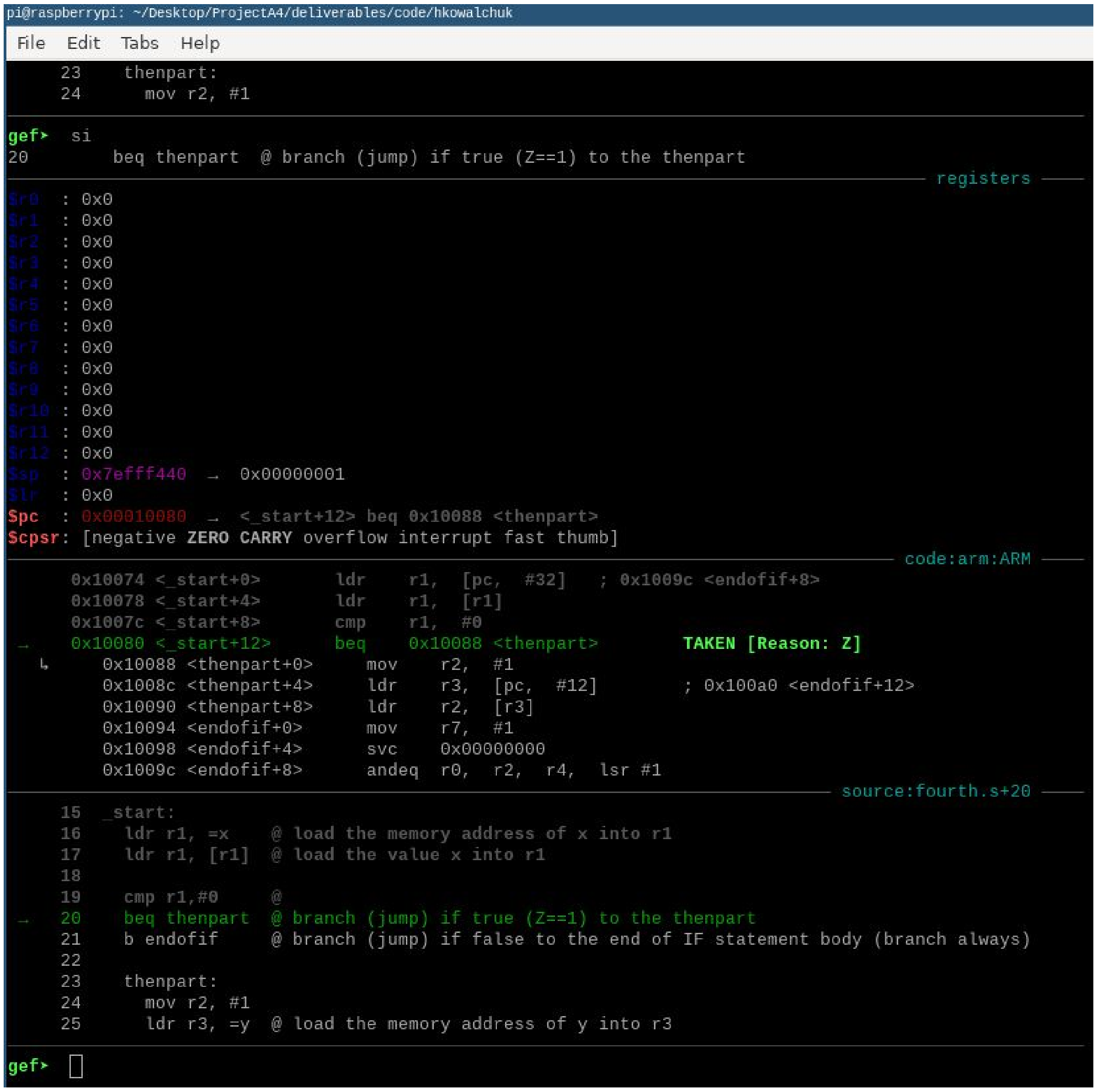
The value of the memory of x is 1 and the value of the Z flag is 1 because the result is stored in the register r1 which is 0.

Hayden Kowalchuk:

Part One:

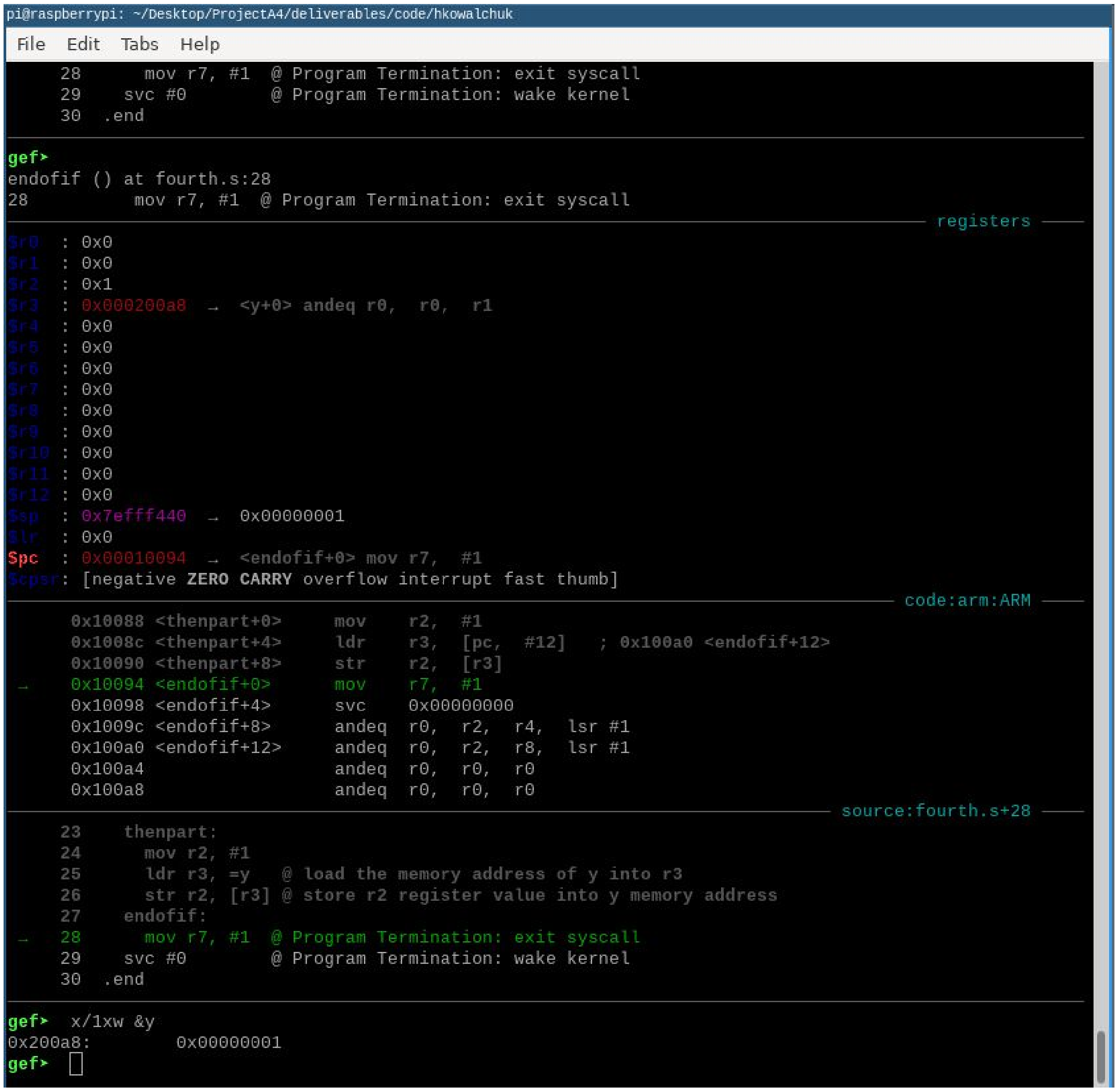
The code that is provided for fourth.s is supposed to demonstrate a simple if clause that sets a value of a variable y if the variable x is equal to 0. Unfortunately in the line that is meant to store the value “ldr r2, [r3]” should be “str r2, [r3]” to have the desired effect. After correcting this line we can trace the program while it is running in GDB, I did this by setting a breakpoint on the compare line and then stepping through each instruction and noting the flags and values of x and y in memory. This kind of if statement to assembly translation is very literal and thus is not the best we could do because of additional logic and instructions, it would be better to invert the check.

This image shows that the jump to “thenpart” is taken when x is 0:


pi@raspberrypi: ~/Desktop/ProjectA4/deliverables/code/hkowalchuk
File Edit Tabs Help

```
     23    thenpart:
     24      mov r2, #1
──────────────────────────────────────────────────────────────────────────────────────
gef➤  si
20        beq thenpart  @ branch (jump) if true (Z==1) to the thenpart
─────────────────────────────────────────────────────────────────────── registers ────
$r0  : 0x0
$r1  : 0x0
$r2  : 0x0
$r3  : 0x0
$r4  : 0x0
$r5  : 0x0
$r6  : 0x0
$r7  : 0x0
$r8  : 0x0
$r9  : 0x0
$r10 : 0x0
$r11 : 0x0
$r12 : 0x0
$sp  : 0x7efff440  →  0x00000001
$lr  : 0x0
$pc  : 0x00010080  →  <_start+12> beq 0x10088 <thenpart>
$cpsr: [negative ZERO CARRY overflow interrupt fast thumb]
─────────────────────────────────────────────────────────────────── code:arm:ARM ────
      0x10074 <_start+0>       ldr    r1,  [pc,  #32]   ; 0x1009c <endofif+8>
      0x10078 <_start+4>       ldr    r1,  [r1]
      0x1007c <_start+8>       cmp    r1,  #0
 →    0x10080 <_start+12>      beq    0x10088 <thenpart>        TAKEN [Reason: Z]
   ↳     0x10088 <thenpart+0>     mov    r2,  #1
         0x1008c <thenpart+4>     ldr    r3,  [pc,  #12]        ; 0x100a0 <endofif+12>
         0x10090 <thenpart+8>     ldr    r2,  [r3]
         0x10094 <endofif+0>      mov    r7,  #1
         0x10098 <endofif+4>      svc    0x00000000
         0x1009c <endofif+8>      andeq  r0,  r2,  r4,  lsr #1
──────────────────────────────────────────────────────────── source:fourth.s+20 ────
     _start:
       ldr r1, =x    @ load the memory address of x into r1
       ldr r1, [r1]  @ load the value x into r1

       cmp r1,#0     @
 →     beq thenpart  @ branch (jump) if true (Z==1) to the thenpart
       b endofif     @ branch (jump) if false to the end of IF statement body (branch always)

       thenpart:
         mov r2, #1
         ldr r3, =y  @ load the memory address of y into r3
──────────────────────────────────────────────────────────────────────────────────────
gef➤  
```

This image is after stepping through the program until just before the termination syscall, it shows that y has successfully been set to 1 when x is 0:


pi@raspberrypi: ~/Desktop/ProjectA4/deliverables/code/hkowalchuk
File Edit Tabs Help

```
     28      mov r7, #1  @ Program Termination: exit syscall
     29    svc #0        @ Program Termination: wake kernel
     30   .end
─────────────────────────────────────────────────────────────────────────────
gef➤
endofif () at fourth.s:28
28          mov r7, #1  @ Program Termination: exit syscall
──────────────────────────────────────────────────────────────── registers ────
$r0  : 0x0
$r1  : 0x0
$r2  : 0x1
$r3  : 0x000200a8  →  <y+0> andeq r0,  r0,  r1
$r4  : 0x0
$r5  : 0x0
$r6  : 0x0
$r7  : 0x0
$r8  : 0x0
$r9  : 0x0
$r10 : 0x0
$r11 : 0x0
$r12 : 0x0
$sp  : 0x7efff440  →  0x00000001
$lr  : 0x0
$pc  : 0x00010094  →  <endofif+0> mov r7,  #1
$cpsr: [negative ZERO CARRY overflow interrupt fast thumb]
─────────────────────────────────────────────────────────────── code:arm:ARM ────
      0x10088 <thenpart+0>     mov    r2,  #1
      0x1008c <thenpart+4>     ldr    r3,  [pc,  #12]   ; 0x100a0 <endofif+12>
      0x10090 <thenpart+8>     str    r2,  [r3]
 →    0x10094 <endofif+0>      mov    r7,  #1
      0x10098 <endofif+4>      svc    0x00000000
      0x1009c <endofif+8>      andeq  r0,  r2,  r4,  lsr #1
      0x100a0 <endofif+12>     andeq  r0,  r2,  r8,  lsr #1
      0x100a4                  andeq  r0,  r0,  r0
      0x100a8                  andeq  r0,  r0,  r0
──────────────────────────────────────────────────────────── source:fourth.s+28 ────
     23    thenpart:
     24      mov r2, #1
     25      ldr r3, =y    @ load the memory address of y into r3
     26      str r2, [r3] @ store r2 register value into y memory address
     27    endofif:
 →   28      mov r7, #1  @ Program Termination: exit syscall
     29    svc #0        @ Program Termination: wake kernel
     30   .end
─────────────────────────────────────────────────────────────────────────────
gef➤  x/1xw &y
0x200a8:        0x00000001
gef➤  
```

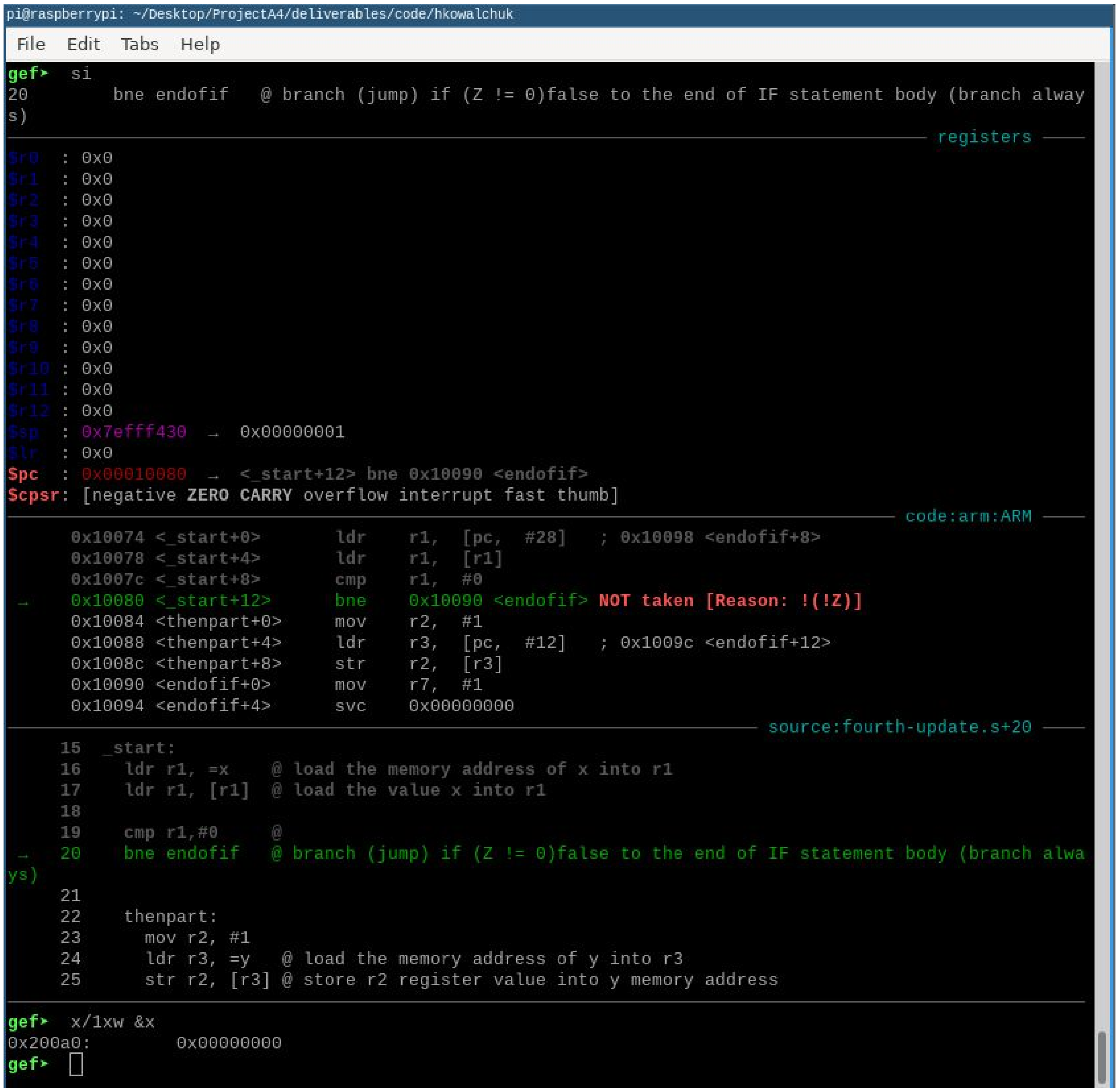
Code:

```
@ Fourth program
@ This program compute the following if statement construct:
@ int x;
@ int y;
@ if(x == 0)
@   y = 1;

.section .data

x: .word 0 @ 32-bit signed integer, you can also use int directive instead
of .word directive
y: .word 0 @ 32-bit signed integer,
.section .text
.globl _start
_start:
  ldr r1, =x     @ load the memory address of x into r1
  ldr r1, [r1]   @ load the value x into r1

  cmp r1,#0      @
  beq thenpart   @ branch (jump) if true (Z==1) to the thenpart
  b endofif       @ branch (jump) if false to the end of IF statement body
(branch always)

  thenpart:
    mov r2, #1
    ldr r3, =y    @ load the memory address of y into r3
    str r2, [r3] @ store r2 register value into y memory address
  endofif:
    mov r7, #1   @ Program Termination: exit syscall
  svc #0         @ Program Termination: wake kernel
.end
```

Part Two:

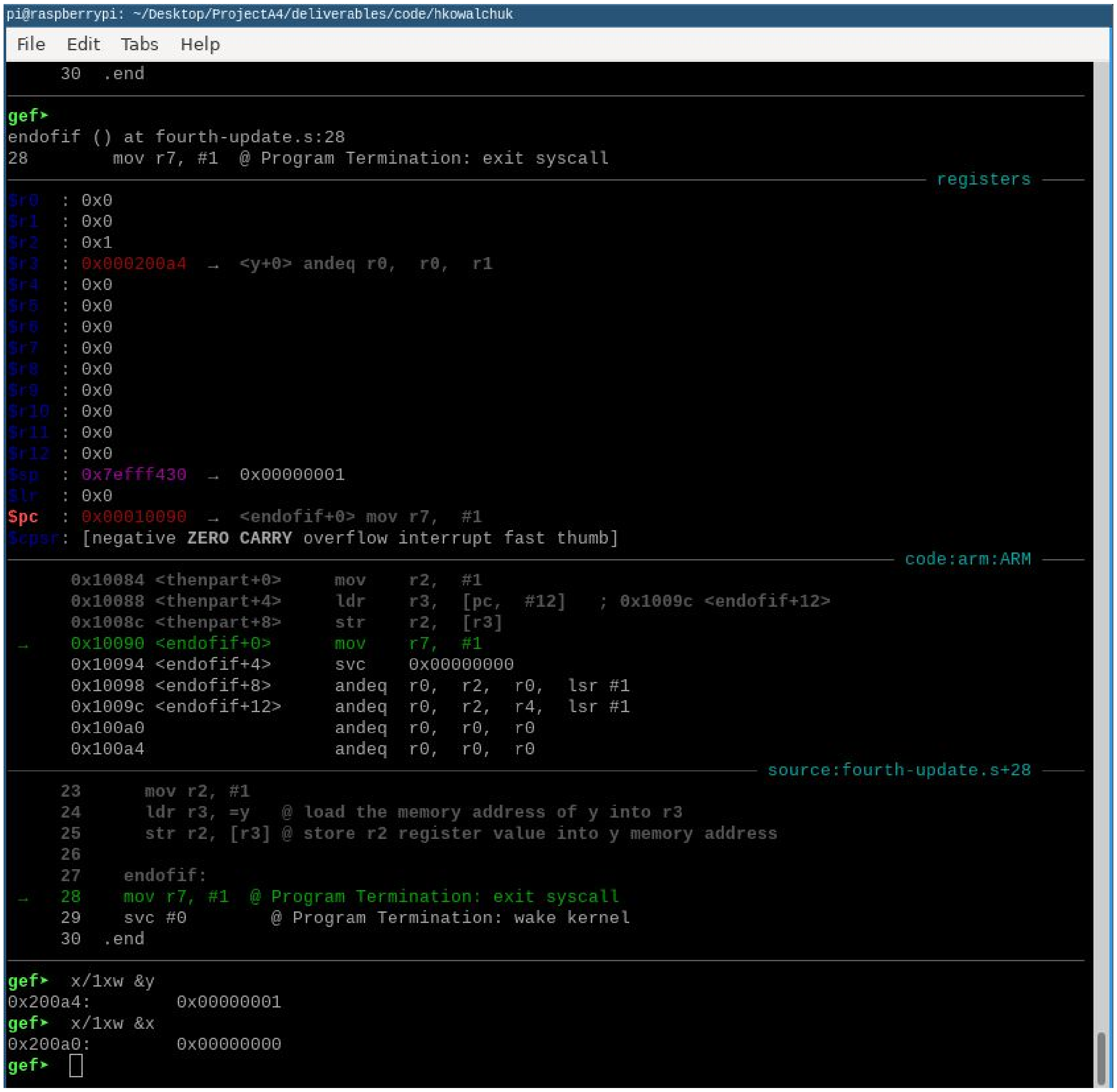
This program is a small extension of the first that addresses the issue of wasting instructions and creating harder to follow code. The only change is that the two branches are moved into a single one and the check is flipped, now it will jump past the inside logic for any value that is not 0. This lets us use a single branch instruction and makes our code cleaner to read, I chose a bne instruction which stands for branch if not equal. The ZERO and CARRY flags are set which means that branch if not equal jump will NOT be taken.

Image showing the new logic that the jump is not taken when x is 0:



```
pi@raspberrypi: ~/Desktop/ProjectA4/deliverables/code/hkowalchuk
File  Edit  Tabs  Help
gef➤  si
20        bne endofif   @ branch (jump) if (Z != 0)false to the end of IF statement body (branch alway
s)
──────────────────────────────────────────────────────────────────────────────── registers ────
$r0  : 0x0
$r1  : 0x0
$r2  : 0x0
$r3  : 0x0
$r4  : 0x0
$r5  : 0x0
$r6  : 0x0
$r7  : 0x0
$r8  : 0x0
$r9  : 0x0
$r10 : 0x0
$r11 : 0x0
$r12 : 0x0
$sp  : 0x7efff430  →  0x00000001
$lr  : 0x0
$pc  : 0x00010080  →  <_start+12> bne 0x10090 <endofif>
$cpsr: [negative ZERO CARRY overflow interrupt fast thumb]
──────────────────────────────────────────────────────────────────────────── code:arm:ARM ────
      0x10074 <_start+0>       ldr    r1,  [pc,  #28]   ; 0x10098 <endofif+8>
      0x10078 <_start+4>       ldr    r1,  [r1]
      0x1007c <_start+8>       cmp    r1,  #0
 →    0x10080 <_start+12>      bne    0x10090 <endofif> NOT taken [Reason: !(!Z)]
      0x10084 <thenpart+0>     mov    r2,  #1
      0x10088 <thenpart+4>     ldr    r3,  [pc,  #12]   ; 0x1009c <endofif+12>
      0x1008c <thenpart+8>     str    r2,  [r3]
      0x10090 <endofif+0>      mov    r7,  #1
      0x10094 <endofif+4>      svc    0x00000000
─────────────────────────────────────────────────────────────── source:fourth-update.s+20 ────
     15  _start:
     16    ldr r1, =x    @ load the memory address of x into r1
     17    ldr r1, [r1]  @ load the value x into r1
     18
     19    cmp r1,#0     @
 →   20    bne endofif   @ branch (jump) if (Z != 0)false to the end of IF statement body (branch alwa
ys)
     21
     22    thenpart:
     23      mov r2, #1
     24      ldr r3, =y   @ load the memory address of y into r3
     25      str r2, [r3] @ store r2 register value into y memory address
──────────────────────────────────────────────────────────────────────────────────────────────
gef➤  x/1xw &x
0x200a0:        0x00000000
gef➤
```

The following image was taken after stepping and is the verification that y is still updated correctly according to x, in this case x=0 and in response y was set to 1


pi@raspberrypi: ~/Desktop/ProjectA4/deliverables/code/hkowalchuk
File Edit Tabs Help

```
    30   .end

gef➤
endofif () at fourth-update.s:28
28          mov r7, #1  @ Program Termination: exit syscall
                                                          registers
$r0   : 0x0
$r1   : 0x0
$r2   : 0x1
$r3   : 0x000200a4  →  <y+0> andeq r0,  r0,  r1
$r4   : 0x0
$r5   : 0x0
$r6   : 0x0
$r7   : 0x0
$r8   : 0x0
$r9   : 0x0
$r10  : 0x0
$r11  : 0x0
$r12  : 0x0
$sp   : 0x7efff430  →  0x00000001
$lr   : 0x0
$pc   : 0x00010090  →  <endofif+0> mov r7,  #1
$cpsr: [negative ZERO CARRY overflow interrupt fast thumb]
                                                          code:arm:ARM
      0x10084 <thenpart+0>     mov    r2,  #1
      0x10088 <thenpart+4>     ldr    r3,  [pc,  #12]   ; 0x1009c <endofif+12>
      0x1008c <thenpart+8>     str    r2,  [r3]
 →    0x10090 <endofif+0>      mov    r7,  #1
      0x10094 <endofif+4>      svc    0x00000000
      0x10098 <endofif+8>      andeq  r0,  r2,  r0,  lsr #1
      0x1009c <endofif+12>     andeq  r0,  r2,  r4,  lsr #1
      0x100a0                  andeq  r0,  r0,  r0
      0x100a4                  andeq  r0,  r0,  r0
                                                          source:fourth-update.s+28
          mov r2, #1
          ldr r3, =y    @ load the memory address of y into r3
          str r2, [r3] @ store r2 register value into y memory address

        endofif:
 →      mov r7, #1  @ Program Termination: exit syscall
          svc #0        @ Program Termination: wake kernel
      .end

gef➤  x/1xw &y
0x200a4:        0x00000001
gef➤  x/1xw &x
0x200a0:        0x00000000
gef➤
```

Code:

```
@ Fourth program
@ This program compute the following if statement construct:
@ int x;
@ int y;
@ if(x == 0)
@    y = 1;

.section .data
```

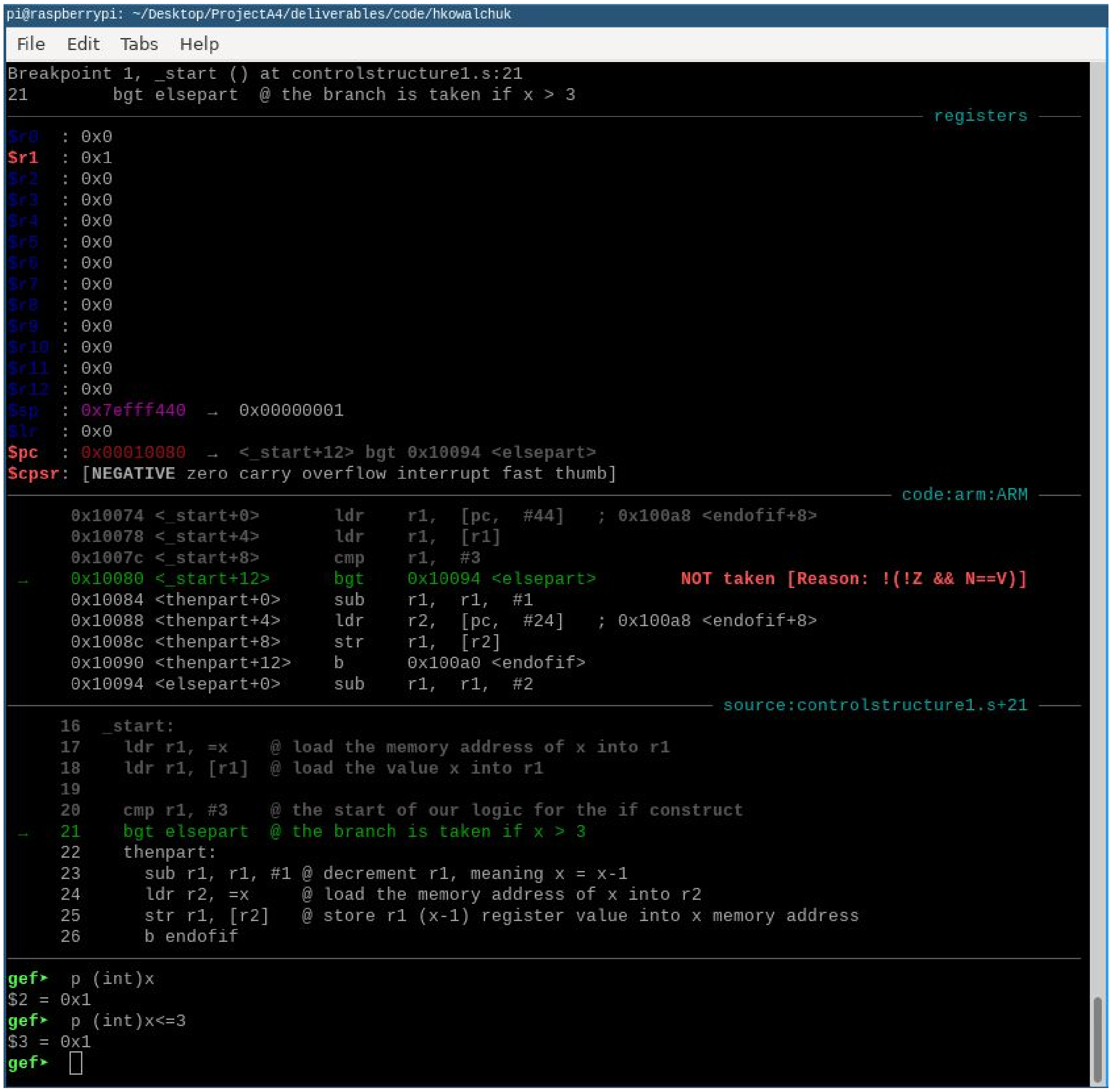
```
x: .word 0 @ 32-bit signed integer, you can also use int directive instead
of .word directive
y: .word 0 @ 32-bit signed integer,

.section .text
.globl _start
_start:
  ldr r1, =x     @ load the memory address of x into r1
  ldr r1, [r1]   @ load the value x into r1

  cmp r1,#0      @
   bne  endofif     @  branch  (jump)  if  (Z  !=  0)  false  to  the  end  of  IF
statement body

  thenpart:
    mov r2, #1
    ldr r3, =y    @ load the memory address of y into r3
    str r2, [r3] @ store r2 register value into y memory address

  endofif:
  mov r7, #1   @ Program Termination: exit syscall
  svc #0       @ Program Termination: wake kernel
.end
```

Part Three:

For this example I first planned out the logic and how it would work on paper, and to me it made the most sense to use the bgt instruction, branch if greater than, for our first logic control. After I had decided on that instruction the rest of the code naturally flowed much like the given pseudo code that we were given to recreate in ARM assembly. The "thenpart" label is the logic for when our if statement is true, x is less than or equal to 3, and it subtracts 1 from x then stores that result back in x, the "elsepart" label is similar in that it subtracts 2 from x and stores that back in x.

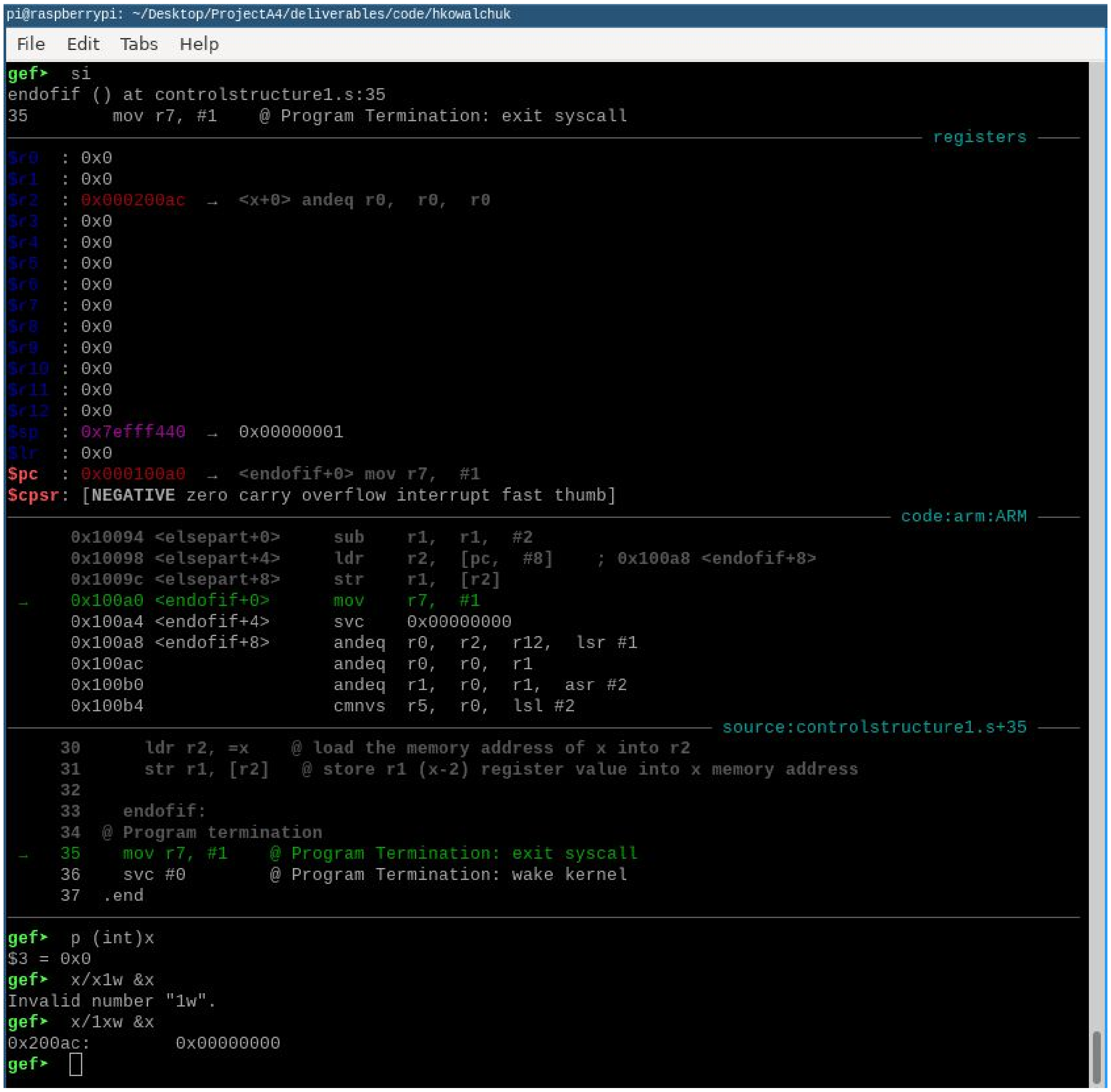
A possible efficiency optimization could be made to always subtract 1 from x, and then change the logic to a single label that we branch to if x is greater than 3 we subtract 1 an additional time. This is not the version of the code I submitted because I feel it goes beyond the scope of the question, but it is presented nonetheless.

Image showing the jump is NOT taken when x <= 3, and then 1 is subtracted from x:



```
pi@raspberrypi: ~/Desktop/ProjectA4/deliverables/code/hkowalchuk
File  Edit  Tabs  Help
Breakpoint 1, _start () at controlstructure1.s:21
21        bgt elsepart  @ the branch is taken if x > 3
──────────────────────────────────────────────────────────────────── registers ────
$r0  : 0x0
$r1  : 0x1
$r2  : 0x0
$r3  : 0x0
$r4  : 0x0
$r5  : 0x0
$r6  : 0x0
$r7  : 0x0
$r8  : 0x0
$r9  : 0x0
$r10 : 0x0
$r11 : 0x0
$r12 : 0x0
$sp  : 0x7efff440  →  0x00000001
$lr  : 0x0
$pc  : 0x00010080  →  <_start+12> bgt 0x10094 <elsepart>
$cpsr: [NEGATIVE zero carry overflow interrupt fast thumb]
────────────────────────────────────────────────────────────────── code:arm:ARM ────
      0x10074 <_start+0>       ldr    r1,  [pc,  #44]   ; 0x100a8 <endofif+8>
      0x10078 <_start+4>       ldr    r1,  [r1]
      0x1007c <_start+8>       cmp    r1,  #3
 →    0x10080 <_start+12>      bgt    0x10094 <elsepart>         NOT taken [Reason: !(!Z && N==V)]
      0x10084 <thenpart+0>     sub    r1,  r1,  #1
      0x10088 <thenpart+4>     ldr    r2,  [pc,  #24]   ; 0x100a8 <endofif+8>
      0x1008c <thenpart+8>     str    r1,  [r2]
      0x10090 <thenpart+12>    b      0x100a0 <endofif>
      0x10094 <elsepart+0>     sub    r1,  r1,  #2
──────────────────────────────────────────────────── source:controlstructure1.s+21 ────
     16  _start:
     17    ldr r1, =x     @ load the memory address of x into r1
     18    ldr r1, [r1]   @ load the value x into r1
     19
     20    cmp r1, #3     @ the start of our logic for the if construct
 →   21    bgt elsepart   @ the branch is taken if x > 3
     22    thenpart:
     23      sub r1, r1, #1 @ decrement r1, meaning x = x-1
     24      ldr r2, =x      @ load the memory address of x into r2
     25      str r1, [r2]    @ store r1 (x-1) register value into x memory address
     26      b endofif
────────────────────────────────────────────────────────────────────────────────────
gef➤  p (int)x
$2 = 0x1
gef➤  p (int)x<=3
$3 = 0x1
gef➤  
```

Final image showing the value of the variable x after the jump has been taken before exit:



```
pi@raspberrypi: ~/Desktop/ProjectA4/deliverables/code/hkowalchuk
File  Edit  Tabs  Help
gef➤  si
endofif () at controlstructure1.s:35
35          mov r7, #1    @ Program Termination: exit syscall
──────────────────────────────────────────────────────── registers ────
$r0  : 0x0
$r1  : 0x0
$r2  : 0x000200ac  →  <x+0> andeq r0,  r0,  r0
$r3  : 0x0
$r4  : 0x0
$r5  : 0x0
$r6  : 0x0
$r7  : 0x0
$r8  : 0x0
$r9  : 0x0
$r10 : 0x0
$r11 : 0x0
$r12 : 0x0
$sp  : 0x7efff440  →  0x00000001
$lr  : 0x0
$pc  : 0x000100a0  →  <endofif+0> mov r7,  #1
$cpsr: [NEGATIVE zero carry overflow interrupt fast thumb]
───────────────────────────────────────────────────── code:arm:ARM ────
      0x10094 <elsepart+0>     sub    r1,  r1,  #2
      0x10098 <elsepart+4>     ldr    r2,  [pc,  #8]    ; 0x100a8 <endofif+8>
      0x1009c <elsepart+8>     str    r1,  [r2]
 →    0x100a0 <endofif+0>      mov    r7,  #1
      0x100a4 <endofif+4>      svc    0x00000000
      0x100a8 <endofif+8>      andeq  r0,  r2,  r12,  lsr #1
      0x100ac                  andeq  r0,  r0,  r1
      0x100b0                  andeq  r1,  r0,  r1,  asr #2
      0x100b4                  cmnvs  r5,  r0,  lsl #2
──────────────────────────────────── source:controlstructure1.s+35 ────
     30      ldr r2, =x    @ load the memory address of x into r2
     31      str r1, [r2]   @ store r1 (x-2) register value into x memory address
     32
     33    endofif:
     34  @ Program termination
 →   35      mov r7, #1     @ Program Termination: exit syscall
     36      svc #0         @ Program Termination: wake kernel
     37    .end
───────────────────────────────────────────────────────────────────────
gef➤  p (int)x
$3 = 0x0
gef➤  x/x1w &x
Invalid number "1w".
gef➤  x/1xw &x
0x200ac:        0x00000000
gef➤  
```

Code:

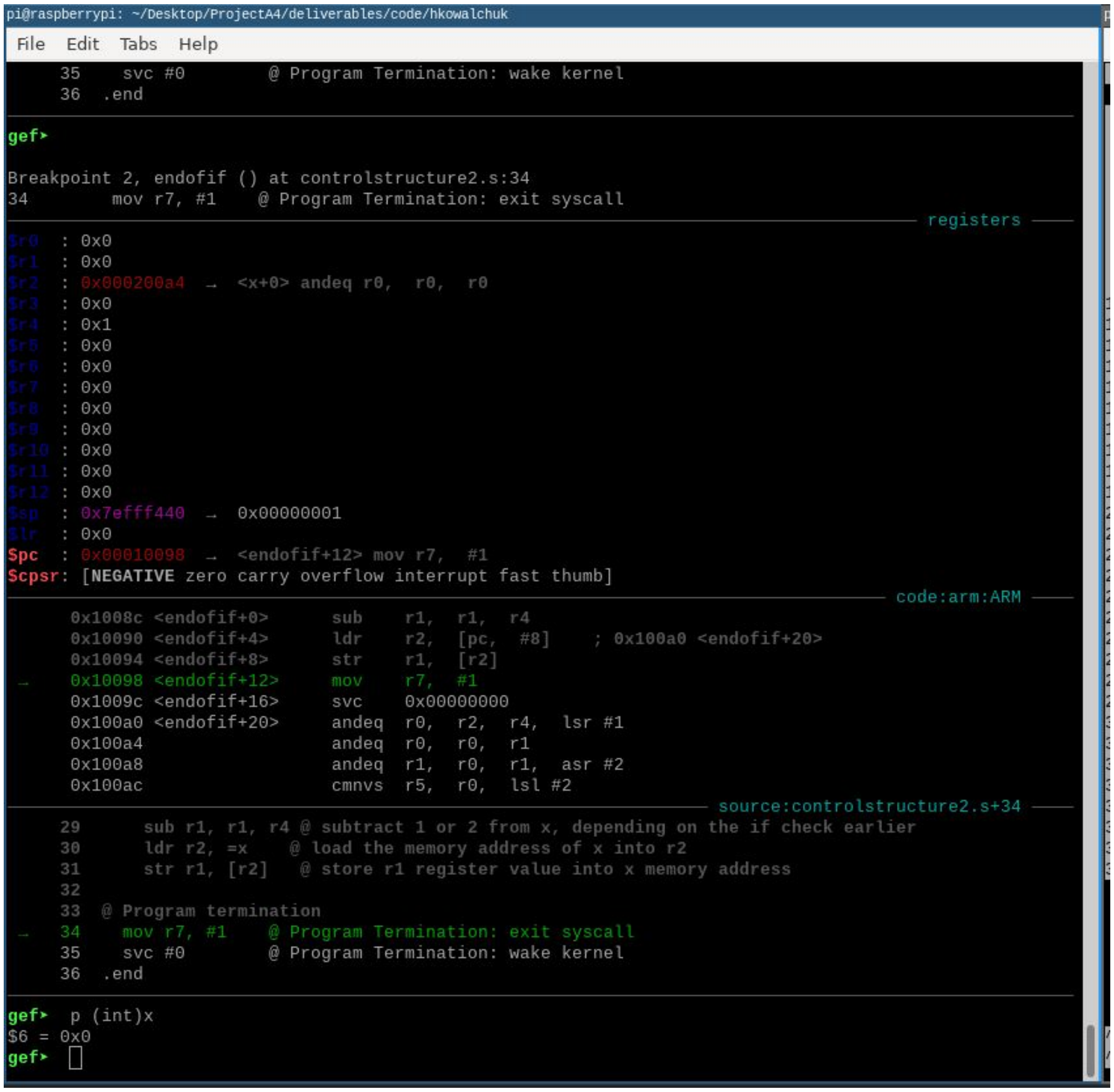
```
@ Control Structure
@ Hayden Kowalchuk (c) 2020
@ This program compute the following if statement construct:
@ int x = 1;
@ if(x <= 3)
@   x = x-1;
@ else
@   x = x-2;

.section .data

x: .int 1   @ 32-bit signed integer
```

```
.section .text
.globl _start
_start:
  ldr r1, =x    @ load the memory address of x into r1
  ldr r1, [r1]  @ load the value x into r1

  cmp r1, #3    @ the start of our logic for the if construct
  bgt elsepart  @ the branch is taken if x > 3
  thenpart:
    sub r1, r1, #1 @ decrement r1, meaning x = x-1
    ldr r2, =x     @ load the memory address of x into r2
    str r1, [r2]   @ store r1 (x-1) register value into x memory address
    b endofif

  elsepart:
    sub r1, r1, #2 @ subtract 2 from x, x = x-2
    ldr r2, =x     @ load the memory address of x into r2
    str r1, [r2]   @ store r1 (x-2) register value into x memory address

  endofif:
@ Program termination
  mov r7, #1    @ Program Termination: exit syscall
  svc #0        @ Program Termination: wake kernel
.end
```

Extra Code, Concerning the possible reduction and optimization discussed earlier:

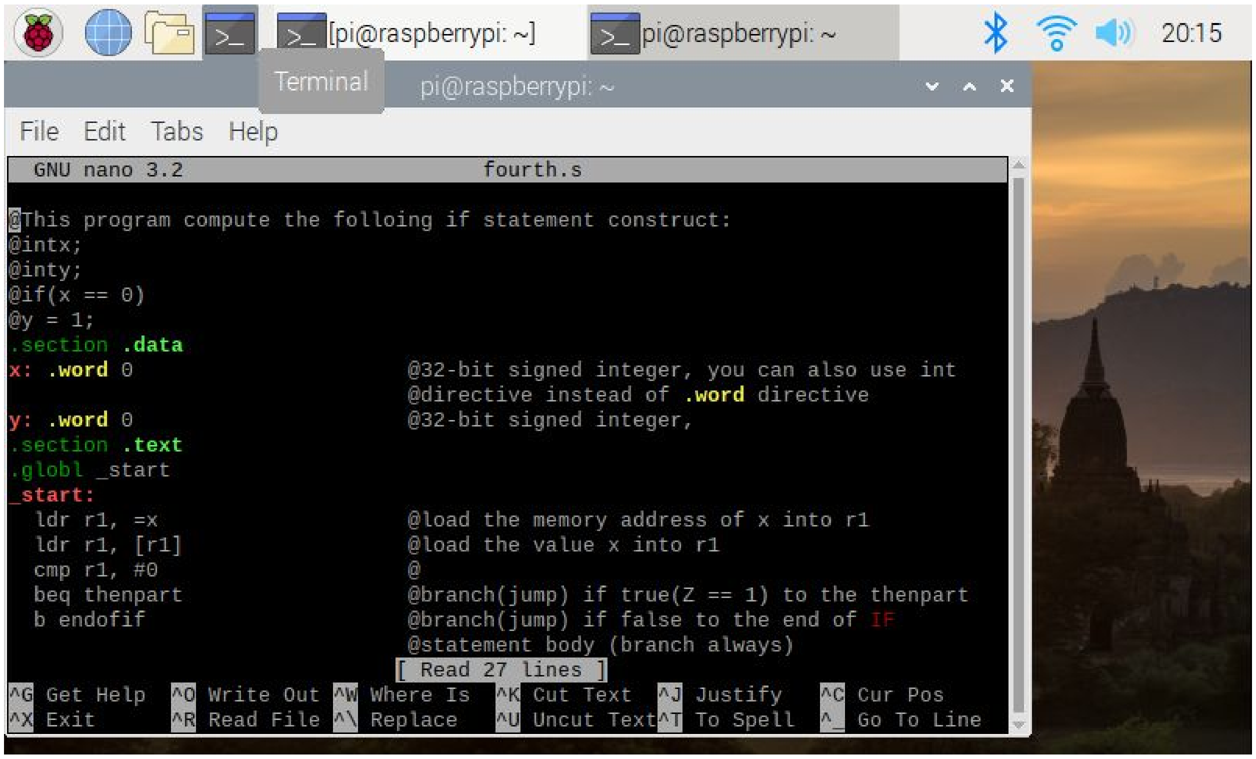
```
@ Control Structure Optimized
@ Hayden Kowalchuk (c) 2020
@ This program compute the following if statement construct:
@ int x = 1;
@ if(x <= 3)
@   x = x-1;
@ else
@   x = x-2;

.section .data

x: .int 1   @ 32-bit signed integer

.section .text
.globl _start
_start:
  ldr r1, =x    @ load the memory address of x into r1
  ldr r1, [r1]  @ load the value x into r1

@!! NEW
  mov r4, #1    @ r4 will be the value we subtract from x, start at 1

  cmp r1, #3    @ the start of our logic for the if construct
  ble endofif   @ the branch is taken if x > 3
 thenpart:
    add r4, r4, #1 @ increment r4, now r4 = 2

 endofif:
    sub r1, r1, r4 @ subtract 1 or 2 from x
    ldr r2, =x     @ load the memory address of x into r2
    str r1, [r2]   @ store r1 register value into x memory address

@ Program termination
  mov r7, #1     @ Program Termination: exit syscall
  svc #0         @ Program Termination: wake kernel
.end
```

Demonstration of this shorter version producing the same result:

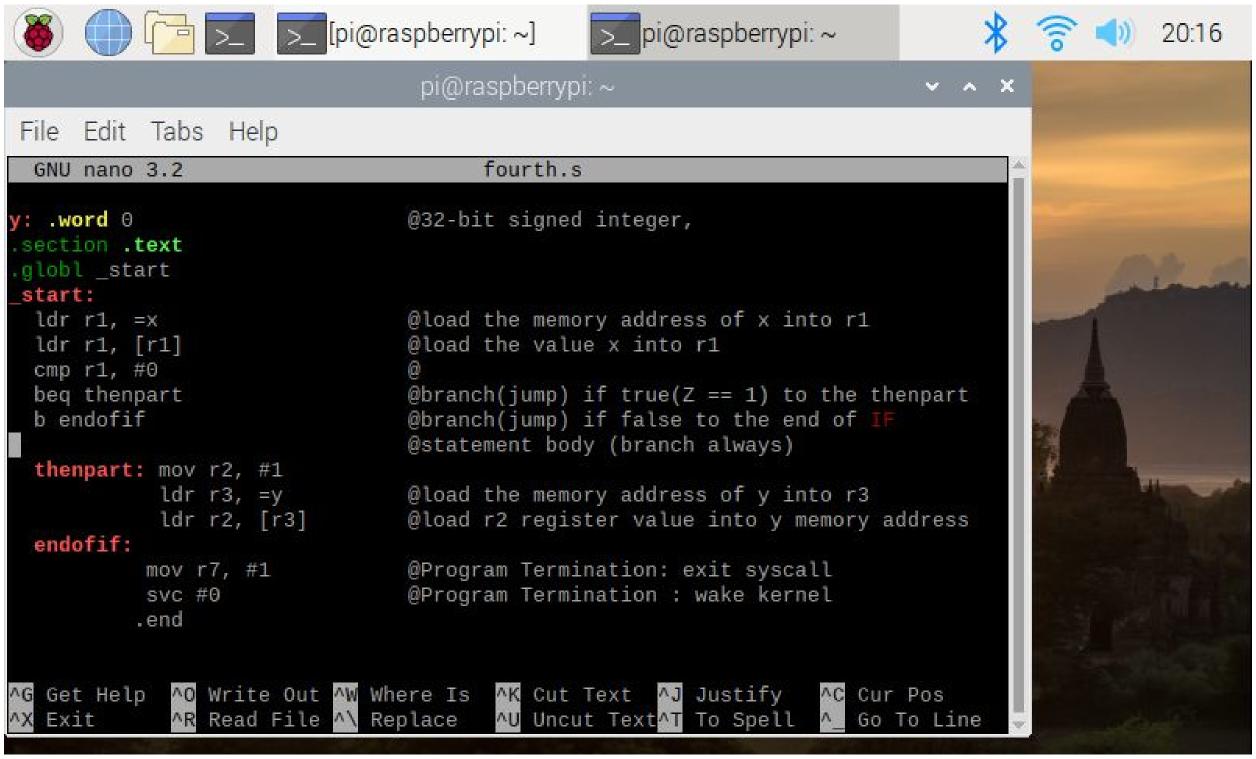


```
pi@raspberrypi: ~/Desktop/ProjectA4/deliverables/code/hkowalchuk
File  Edit  Tabs  Help
     35    svc #0        @ Program Termination: wake kernel
     36   .end
─────────────────────────────────────────────────────────────────────
gef▸

Breakpoint 2, endofif () at controlstructure2.s:34
34        mov r7, #1    @ Program Termination: exit syscall
──────────────────────────────────────────────────────── registers ──
$r0  : 0x0
$r1  : 0x0
$r2  : 0x000200a4  →  <x+0> andeq r0,  r0,  r0
$r3  : 0x0
$r4  : 0x1
$r5  : 0x0
$r6  : 0x0
$r7  : 0x0
$r8  : 0x0
$r9  : 0x0
$r10 : 0x0
$r11 : 0x0
$r12 : 0x0
$sp  : 0x7efff440  →  0x00000001
$lr  : 0x0
$pc  : 0x00010098  →  <endofif+12> mov r7,  #1
$cpsr: [NEGATIVE zero carry overflow interrupt fast thumb]
──────────────────────────────────────────────────── code:arm:ARM ──
      0x1008c <endofif+0>      sub    r1,  r1,  r4
      0x10090 <endofif+4>      ldr    r2,  [pc,  #8]    ; 0x100a0 <endofif+20>
      0x10094 <endofif+8>      str    r1,  [r2]
 →    0x10098 <endofif+12>     mov    r7,  #1
      0x1009c <endofif+16>     svc    0x00000000
      0x100a0 <endofif+20>     andeq  r0,  r2,  r4,  lsr #1
      0x100a4                  andeq  r0,  r0,  r1
      0x100a8                  andeq  r1,  r0,  r1,  asr #2
      0x100ac                  cmnvs  r5,  r0,  lsl #2
─────────────────────────────────────── source:controlstructure2.s+34 ──
     29      sub r1, r1, r4 @ subtract 1 or 2 from x, depending on the if check earlier
     30      ldr r2, =x     @ load the memory address of x into r2
     31      str r1, [r2]    @ store r1 register value into x memory address
     32
     33   @ Program termination
 →   34    mov r7, #1     @ Program Termination: exit syscall
     35    svc #0         @ Program Termination: wake kernel
     36   .end
─────────────────────────────────────────────────────────────────────
gef▸  p (int)x
$6 = 0x0
gef▸
```

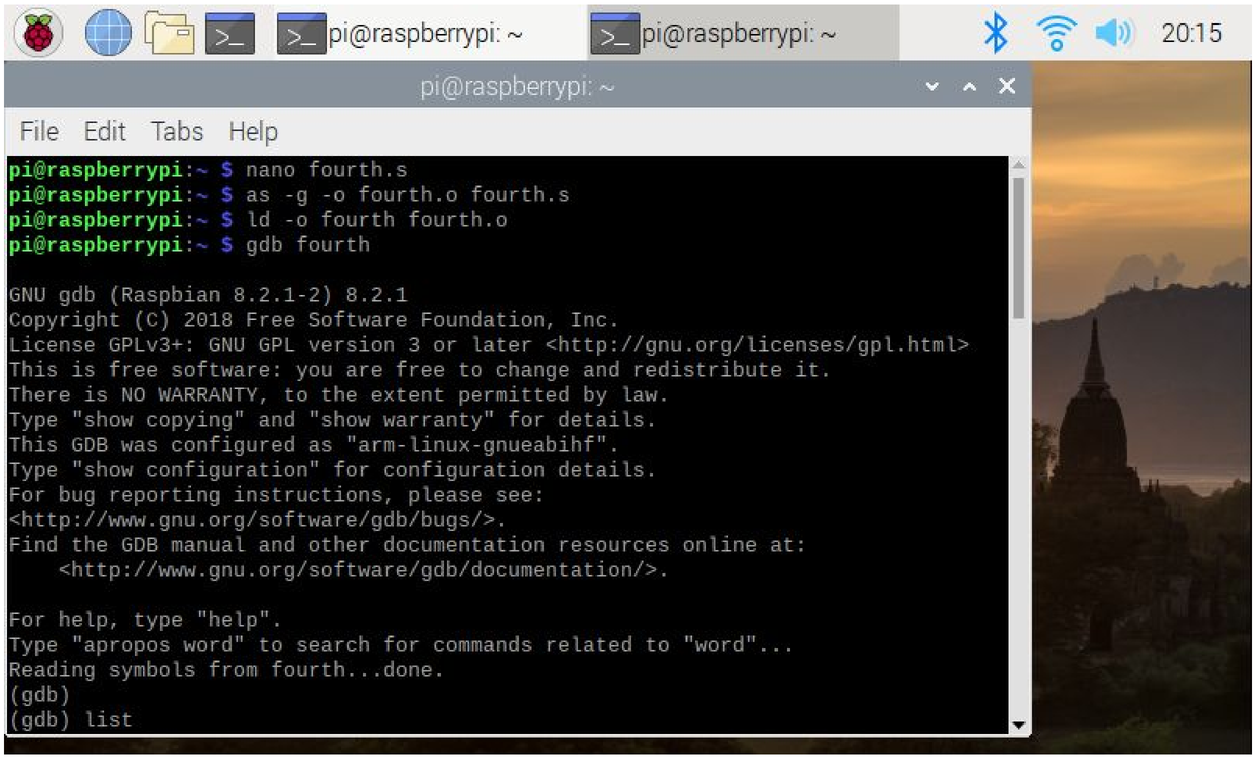
Vera Barnor Agyiri:
Part One:



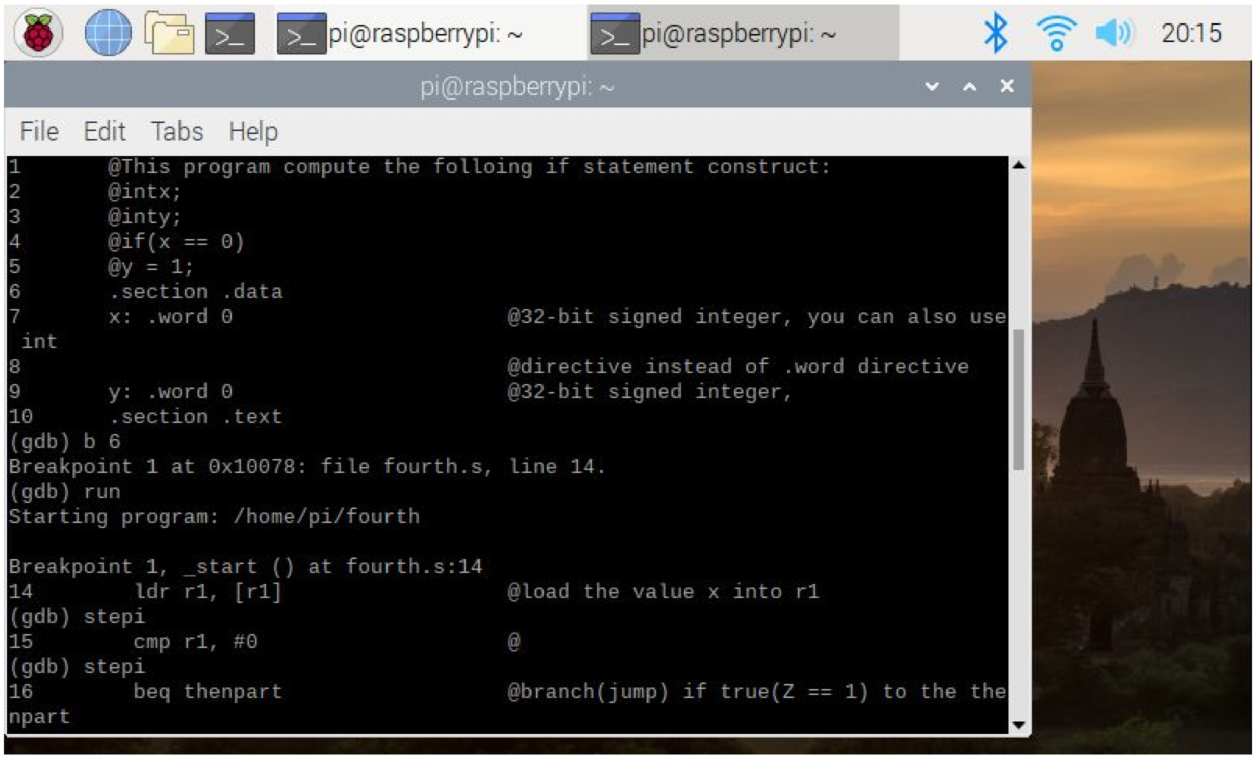
```
  GNU nano 3.2                          fourth.s

@This program compute the folloing if statement construct:
@intx;
@inty;
@if(x == 0)
@y = 1;
.section .data
x: .word 0                      @32-bit signed integer, you can also use int
                                @directive instead of .word directive
y: .word 0                      @32-bit signed integer,
.section .text
.globl _start
_start:
  ldr r1, =x                    @load the memory address of x into r1
  ldr r1, [r1]                  @load the value x into r1
  cmp r1, #0                    @
  beq thenpart                  @branch(jump) if true(Z == 1) to the thenpart
  b endofif                     @branch(jump) if false to the end of IF
                                @statement body (branch always)
                               [ Read 27 lines ]
^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
```



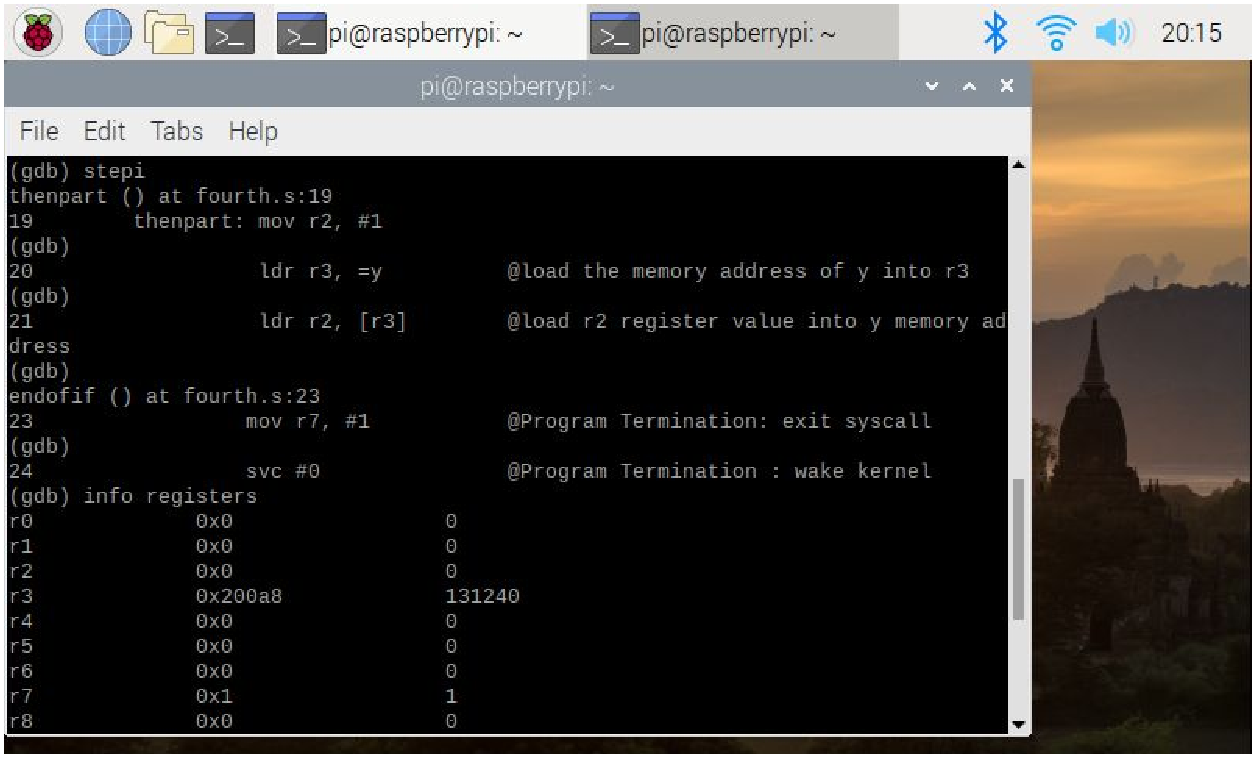
```
  GNU nano 3.2                          fourth.s

y: .word 0                      @32-bit signed integer,
.section .text
.globl _start
_start:
  ldr r1, =x                    @load the memory address of x into r1
  ldr r1, [r1]                  @load the value x into r1
  cmp r1, #0                    @
  beq thenpart                  @branch(jump) if true(Z == 1) to the thenpart
  b endofif                     @branch(jump) if false to the end of IF
                                @statement body (branch always)
  thenpart: mov r2, #1
            ldr r3, =y          @load the memory address of y into r3
            ldr r2, [r3]        @load r2 register value into y memory address
  endofif:
           mov r7, #1           @Program Termination: exit syscall
           svc #0               @Program Termination : wake kernel
          .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
```



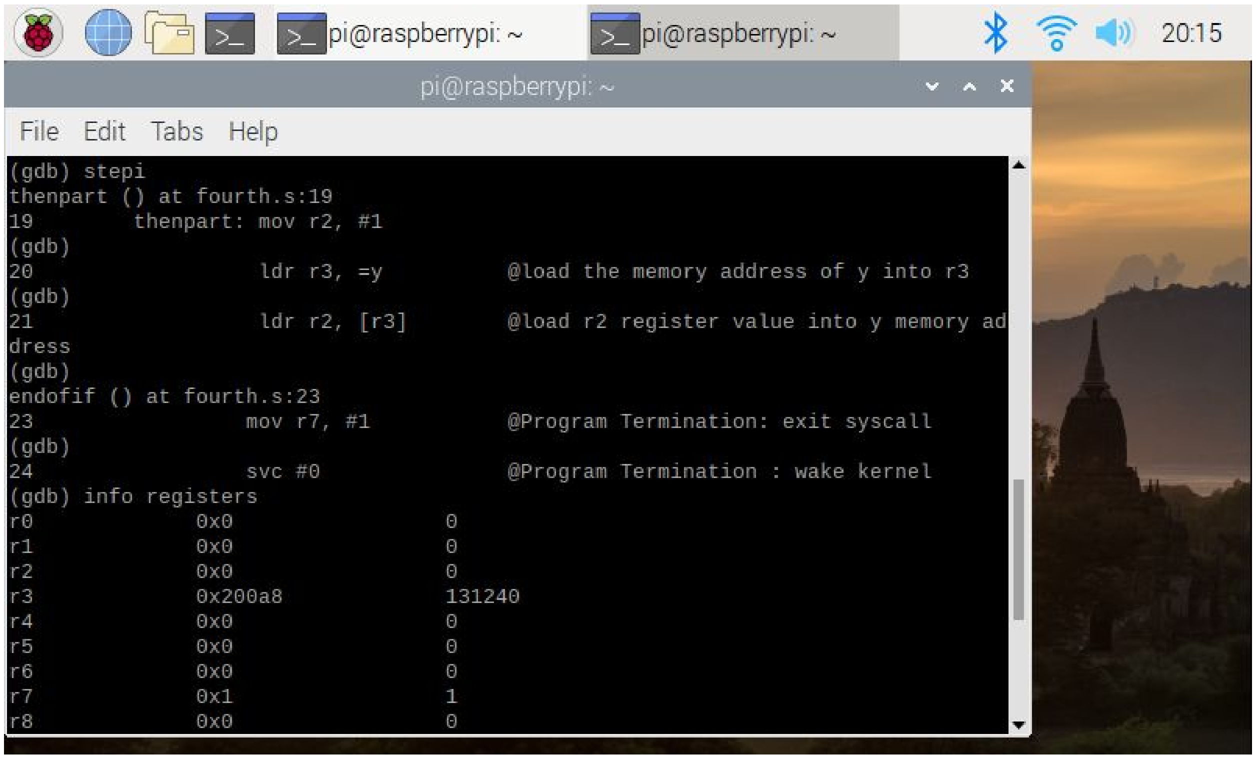
```
pi@raspberrypi:~ $ nano fourth.s
pi@raspberrypi:~ $ as -g -o fourth.o fourth.s
pi@raspberrypi:~ $ ld -o fourth fourth.o
pi@raspberrypi:~ $ gdb fourth

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 fourth...done.
(gdb)
(gdb) list
```



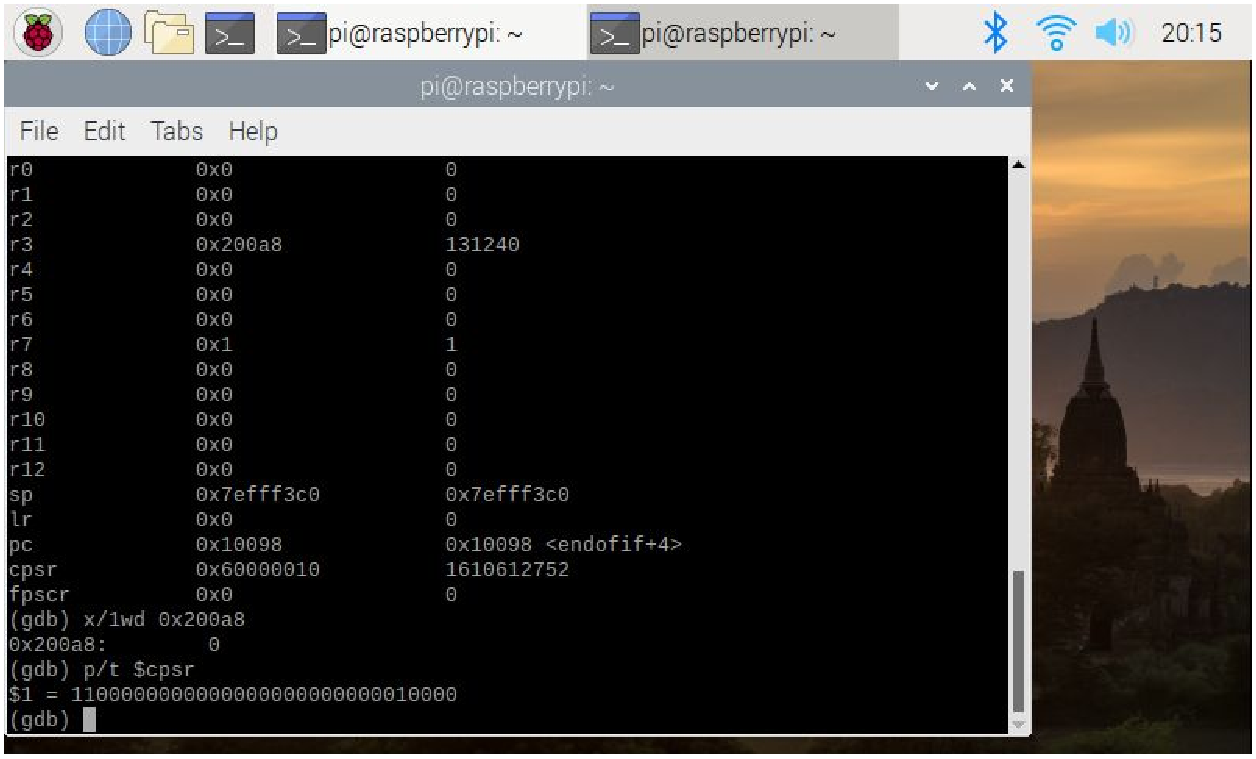
```
1       @This program compute the folloing if statement construct:
2       @intx;
3       @inty;
4       @if(x == 0)
5       @y = 1;
6       .section .data
7       x: .word 0                      @32-bit signed integer, you can also use
 int
8                                       @directive instead of .word directive
9       y: .word 0                      @32-bit signed integer,
10      .section .text
(gdb) b 6
Breakpoint 1 at 0x10078: file fourth.s, line 14.
(gdb) run
Starting program: /home/pi/fourth

Breakpoint 1, _start () at fourth.s:14
14        ldr r1, [r1]                  @load the value x into r1
(gdb) stepi
15        cmp r1, #0                    @
(gdb) stepi
16        beq thenpart                  @branch(jump) if true(Z == 1) to the the
npart
```



```
(gdb) stepi
thenpart () at fourth.s:19
19          thenpart: mov r2, #1
(gdb)
20                    ldr r3, =y          @load the memory address of y into r3
(gdb)
21                    ldr r2, [r3]        @load r2 register value into y memory ad
dress
(gdb)
endofif () at fourth.s:23
23                   mov r7, #1          @Program Termination: exit syscall
(gdb)
24                   svc #0              @Program Termination : wake kernel
(gdb) info registers
r0             0x0                 0
r1             0x0                 0
r2             0x0                 0
r3             0x200a8             131240
r4             0x0                 0
r5             0x0                 0
r6             0x0                 0
r7             0x1                 1
r8             0x0                 0
```

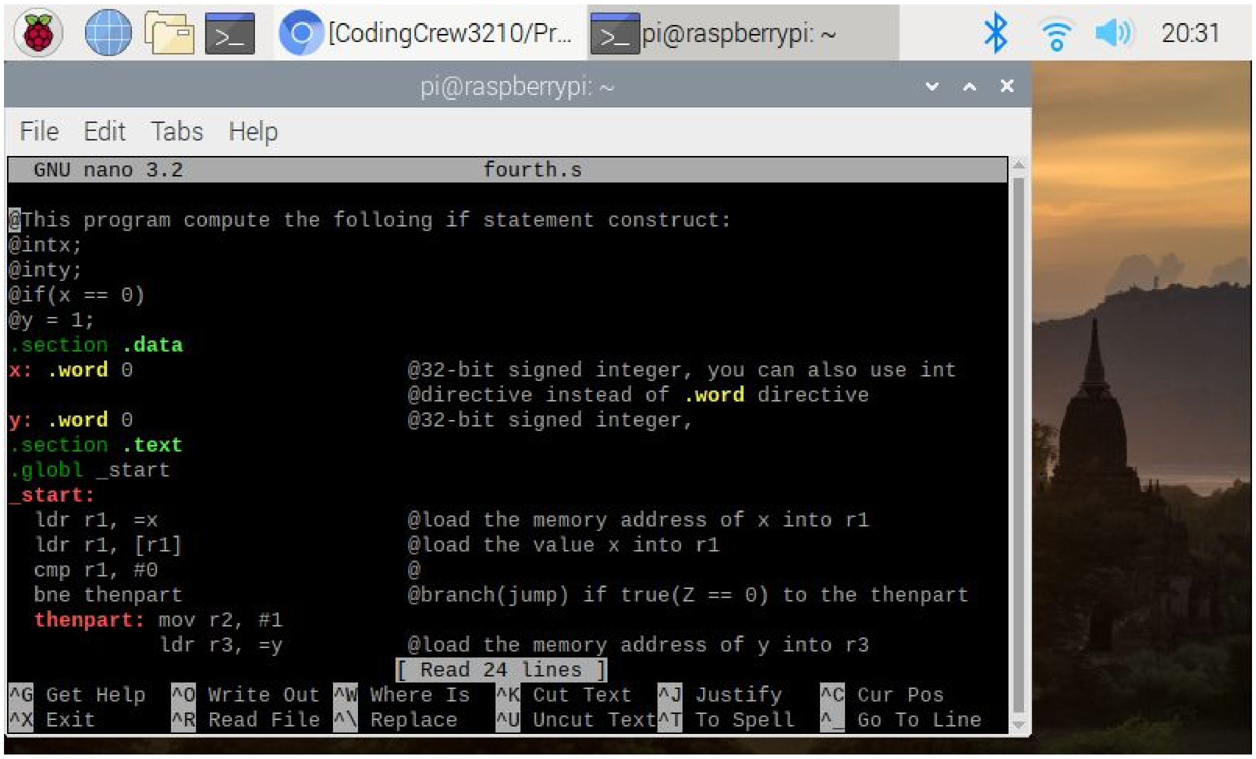


```
(gdb) stepi
thenpart () at fourth.s:19
19          thenpart: mov r2, #1
(gdb)
20                    ldr r3, =y          @load the memory address of y into r3
(gdb)
21                    ldr r2, [r3]        @load r2 register value into y memory ad
dress
(gdb)
endofif () at fourth.s:23
23                   mov r7, #1          @Program Termination: exit syscall
(gdb)
24                   svc #0              @Program Termination : wake kernel
(gdb) info registers
r0             0x0                 0
r1             0x0                 0
r2             0x0                 0
r3             0x200a8             131240
r4             0x0                 0
r5             0x0                 0
r6             0x0                 0
r7             0x1                 1
r8             0x0                 0
```

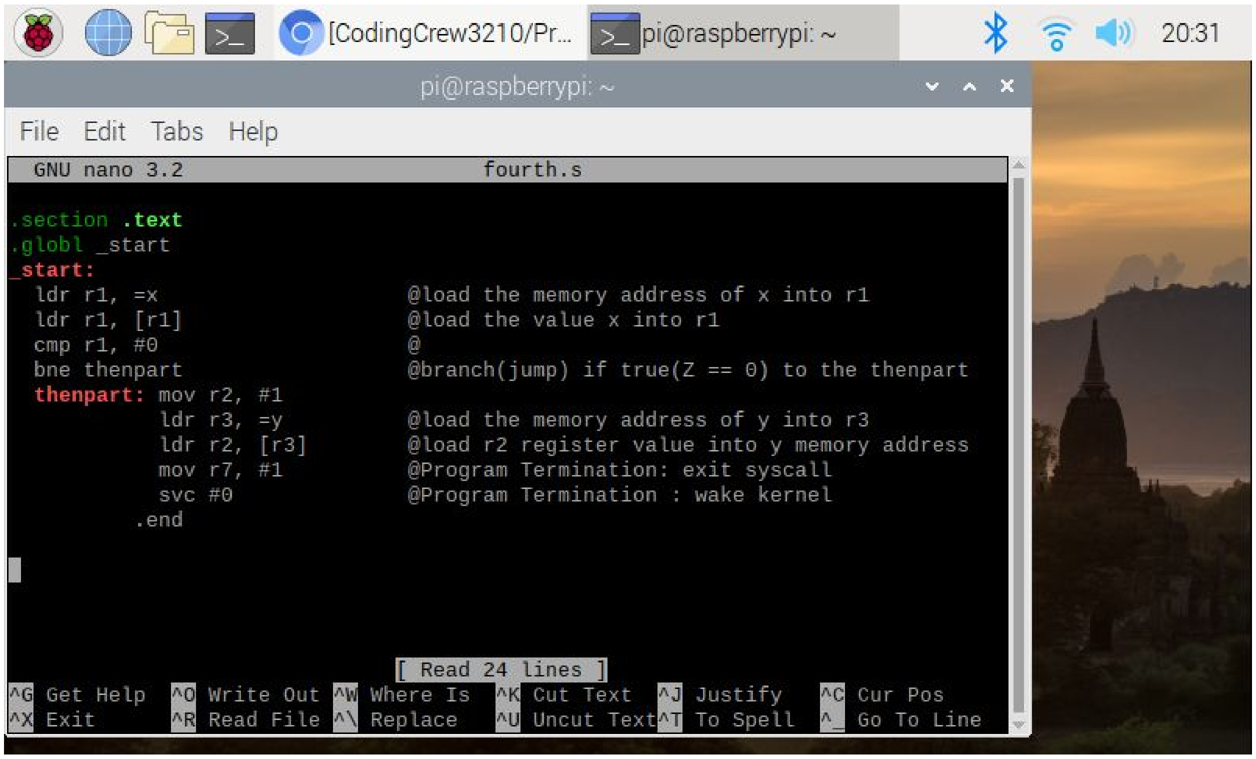


```
r0             0x0                 0
r1             0x0                 0
r2             0x0                 0
r3             0x200a8             131240
r4             0x0                 0
r5             0x0                 0
r6             0x0                 0
r7             0x1                 1
r8             0x0                 0
r9             0x0                 0
r10            0x0                 0
r11            0x0                 0
r12            0x0                 0
sp             0x7efff3c0          0x7efff3c0
lr             0x0                 0
pc             0x10098             0x10098 <endofif+4>
cpsr           0x60000010          1610612752
fpscr          0x0                 0
(gdb) x/1wd 0x200a8
0x200a8:        0
(gdb) p/t $cpsr
$1 = 1100000000000000000000000010000
(gdb) 
```

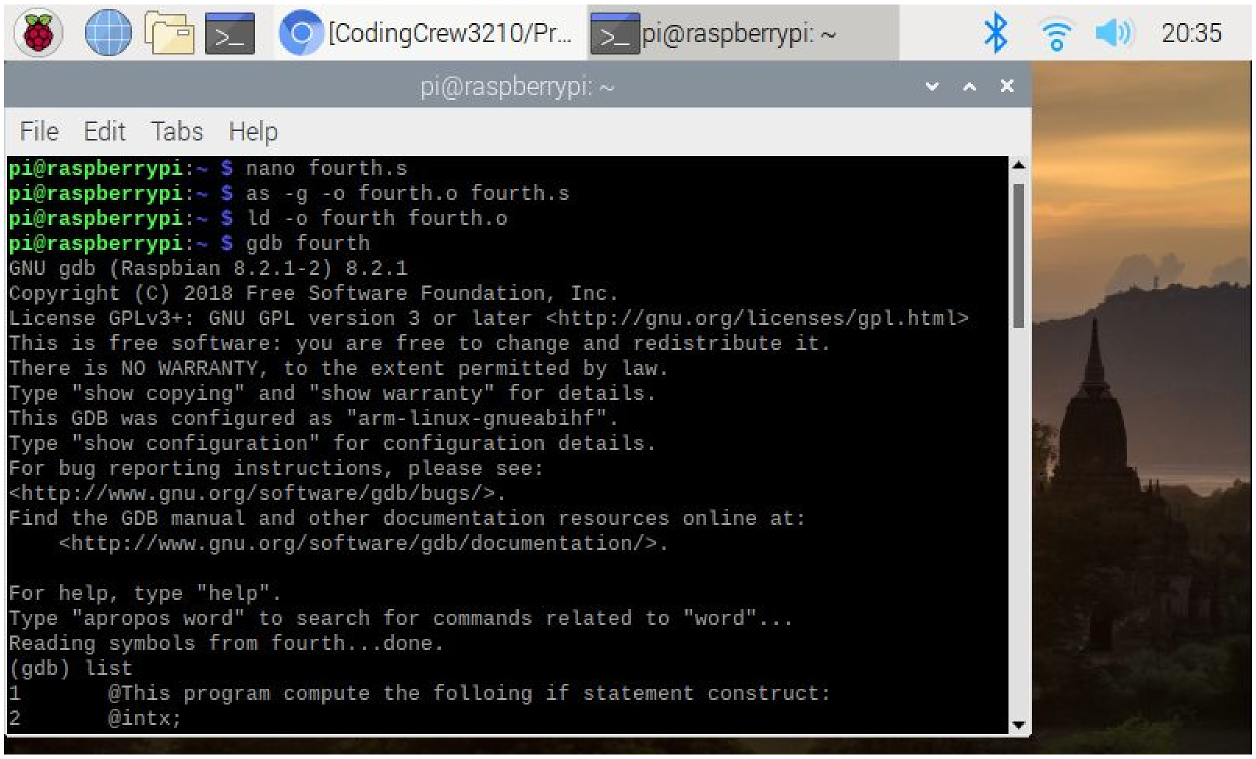
Part Two:



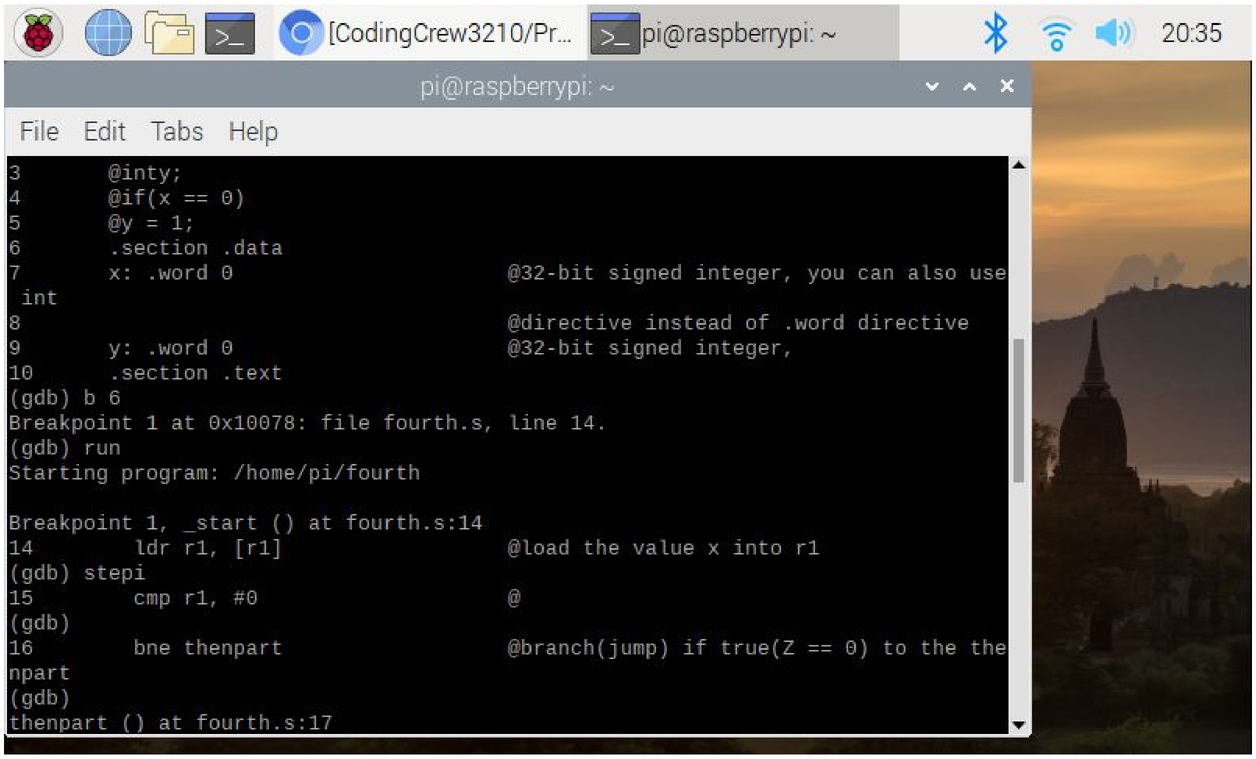
```
  GNU nano 3.2                        fourth.s

@This program compute the folloing if statement construct:
@intx;
@inty;
@if(x == 0)
@y = 1;
.section .data
x: .word 0                      @32-bit signed integer, you can also use int
                                @directive instead of .word directive
y: .word 0                      @32-bit signed integer,
.section .text
.globl _start
_start:
  ldr r1, =x                    @load the memory address of x into r1
  ldr r1, [r1]                  @load the value x into r1
  cmp r1, #0                    @
  bne thenpart                  @branch(jump) if true(Z == 0) to the thenpart
  thenpart: mov r2, #1
            ldr r3, =y          @load the memory address of y into r3
                               [ Read 24 lines ]
^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
```



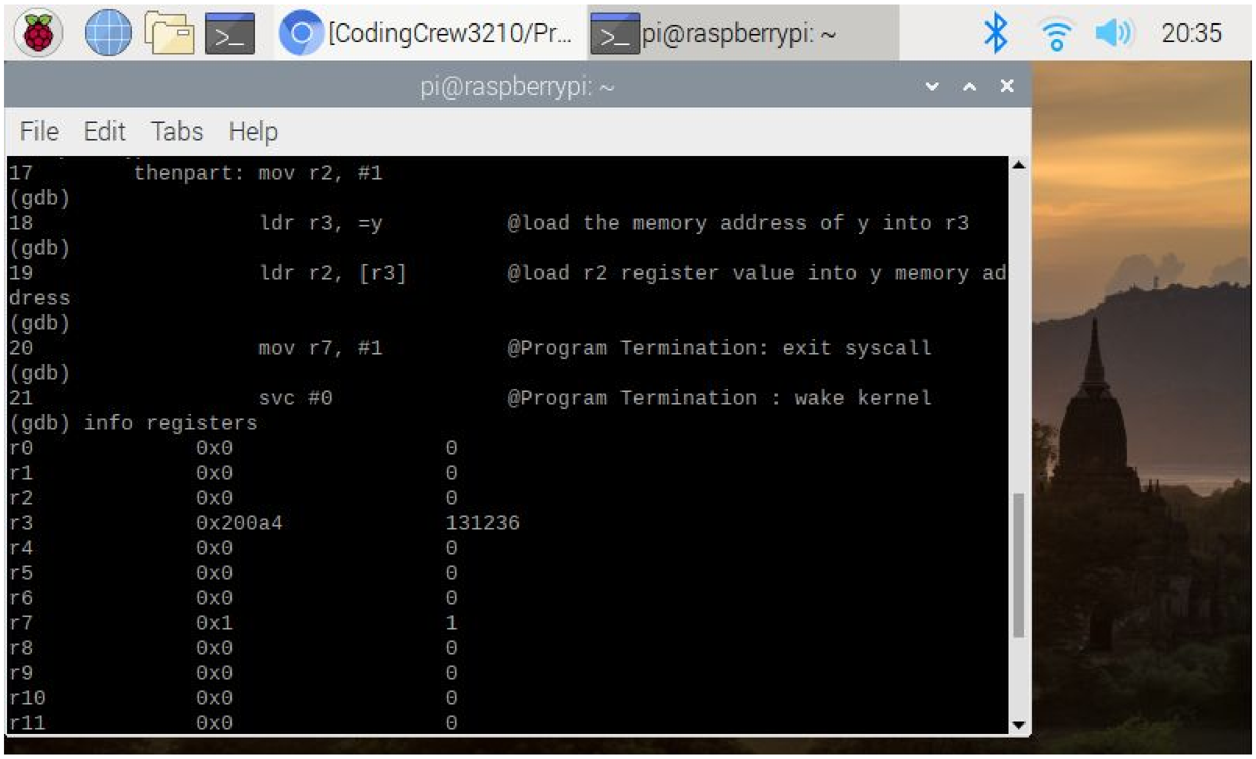
```
  GNU nano 3.2                        fourth.s

.section .text
.globl _start
_start:
  ldr r1, =x                      @load the memory address of x into r1
  ldr r1, [r1]                    @load the value x into r1
  cmp r1, #0                      @
  bne thenpart                    @branch(jump) if true(Z == 0) to the thenpart
  thenpart: mov r2, #1
            ldr r3, =y            @load the memory address of y into r3
            ldr r2, [r3]          @load r2 register value into y memory address
            mov r7, #1            @Program Termination: exit syscall
            svc #0                @Program Termination : wake kernel
          .end


                               [ Read 24 lines ]
^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
```



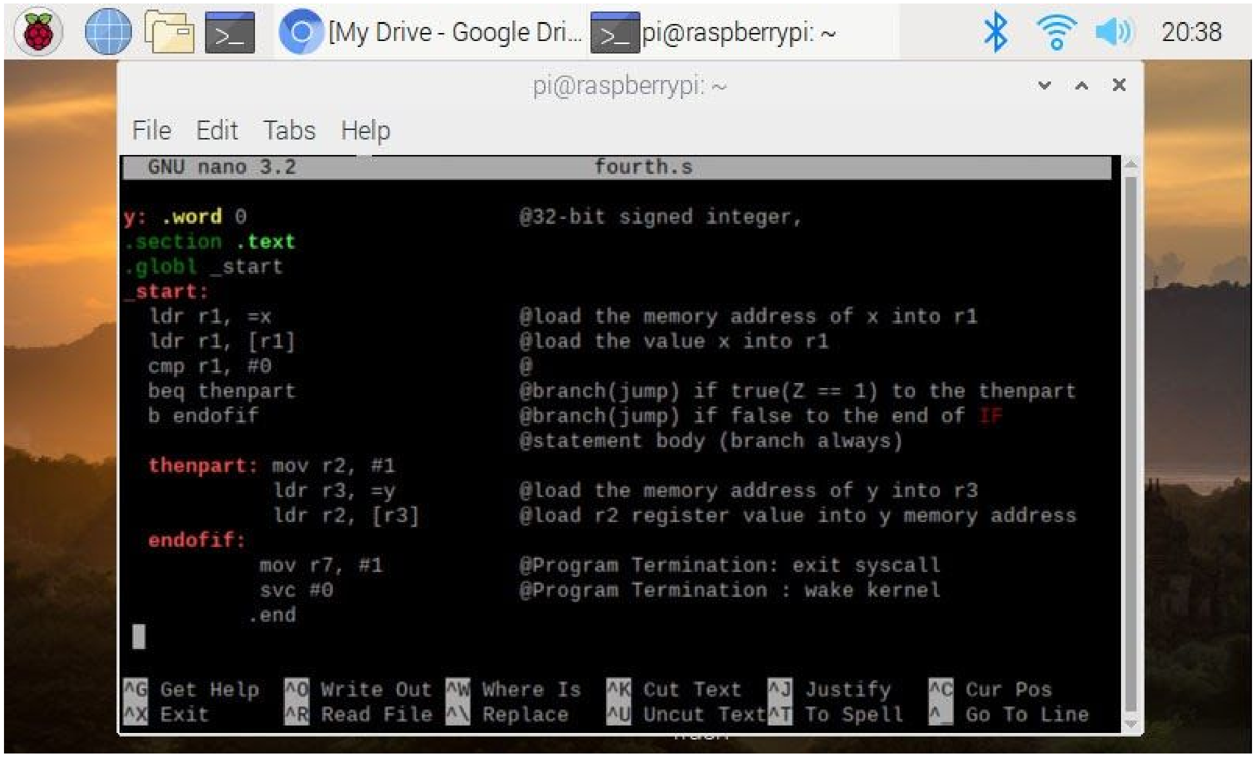
```
pi@raspberrypi:~ $ nano fourth.s
pi@raspberrypi:~ $ as -g -o fourth.o fourth.s
pi@raspberrypi:~ $ ld -o fourth fourth.o
pi@raspberrypi:~ $ gdb fourth
GNU gdb (Raspbian 8.2.1-2) 8.2.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
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 fourth...done.
(gdb) list
1       @This program compute the folloing if statement construct:
2       @intx;
```



```
3       @inty;
4       @if(x == 0)
5       @y = 1;
6       .section .data
7       x: .word 0                      @32-bit signed integer, you can also use
 int
8                                       @directive instead of .word directive
9       y: .word 0                      @32-bit signed integer,
10      .section .text
(gdb) b 6
Breakpoint 1 at 0x10078: file fourth.s, line 14.
(gdb) run
Starting program: /home/pi/fourth

Breakpoint 1, _start () at fourth.s:14
14        ldr r1, [r1]                  @load the value x into r1
(gdb) stepi
15        cmp r1, #0                    @
(gdb)
16        bne thenpart                  @branch(jump) if true(Z == 0) to the the
npart
(gdb)
thenpart () at fourth.s:17
```



```
17        thenpart: mov r2, #1
(gdb)
18                  ldr r3, =y          @load the memory address of y into r3
(gdb)
19                  ldr r2, [r3]        @load r2 register value into y memory ad
dress
(gdb)
20                  mov r7, #1          @Program Termination: exit syscall
(gdb)
21                  svc #0              @Program Termination : wake kernel
(gdb) info registers
r0             0x0                 0
r1             0x0                 0
r2             0x0                 0
r3             0x200a4             131236
r4             0x0                 0
r5             0x0                 0
r6             0x0                 0
r7             0x1                 1
r8             0x0                 0
r9             0x0                 0
r10            0x0                 0
r11            0x0                 0
```



```
  GNU nano 3.2                          fourth.s

y: .word 0                              @32-bit signed integer,
.section .text
.globl _start
_start:
  ldr r1, =x                            @load the memory address of x into r1
  ldr r1, [r1]                          @load the value x into r1
  cmp r1, #0                            @
  beq thenpart                          @branch(jump) if true(Z == 1) to the thenpart
  b endofif                             @branch(jump) if false to the end of IF
                                        @statement body (branch always)
  thenpart: mov r2, #1
            ldr r3, =y                  @load the memory address of y into r3
            ldr r2, [r3]                @load r2 register value into y memory address
  endofif:
           mov r7, #1                   @Program Termination: exit syscall
           svc #0                       @Program Termination : wake kernel
          .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
```

<u>Part Three:</u>

Mohammad Munsur (Rohan):
Part One:

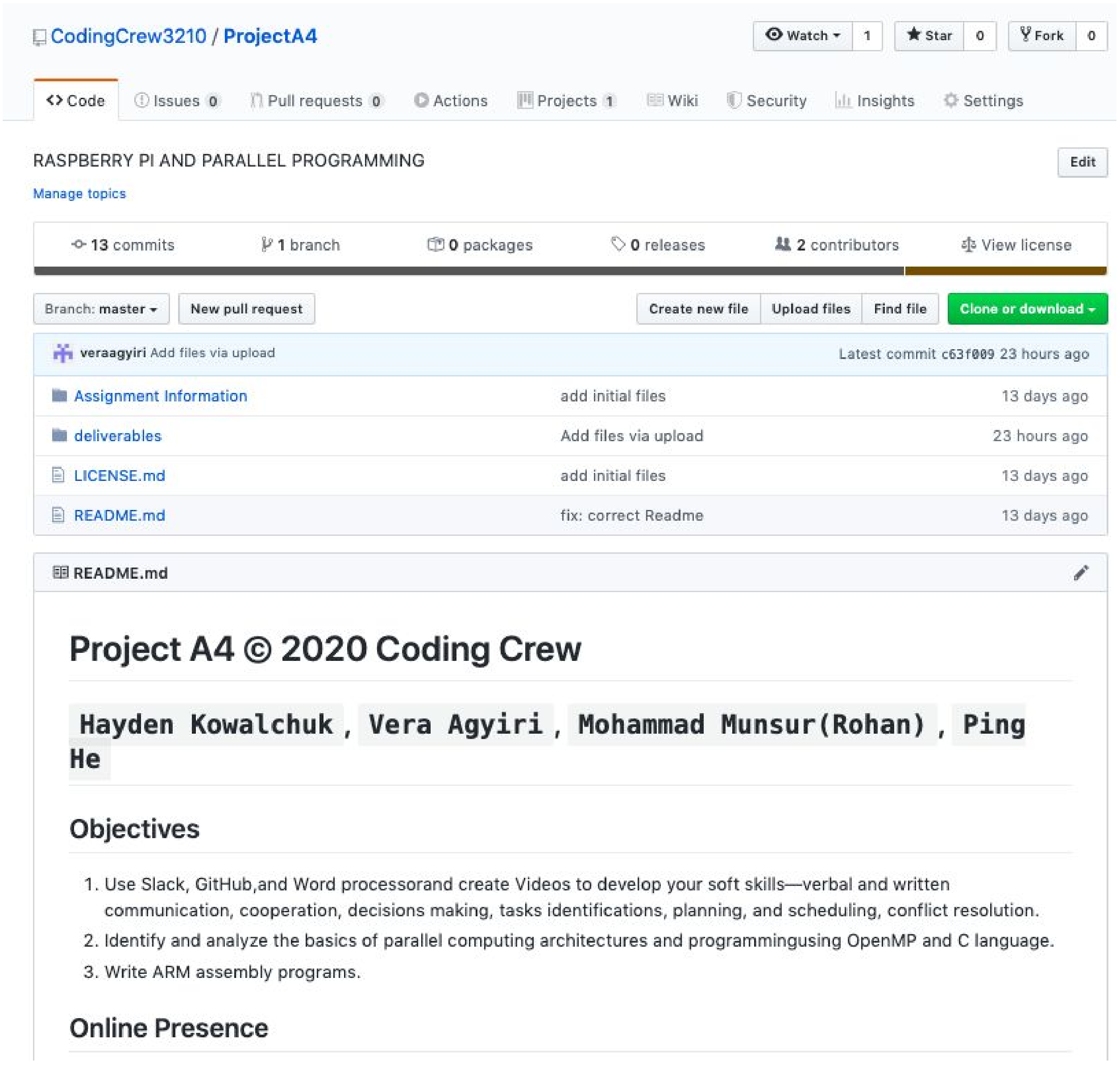
fourth.s program's se that sets a value of a variable y if the variable x is equal to 0.

Part Two:

Part Three:

Appendix

Screenshot of Github:



RASPBERRY PI AND PARALLEL PROGRAMMING

veraagyiri Add files via upload — Latest commit c63f009 23 hours ago

Assignment Information	add initial files	13 days ago
deliverables	Add files via upload	23 hours ago
LICENSE.md	add initial files	13 days ago
README.md	fix: correct Readme	13 days ago

README.md

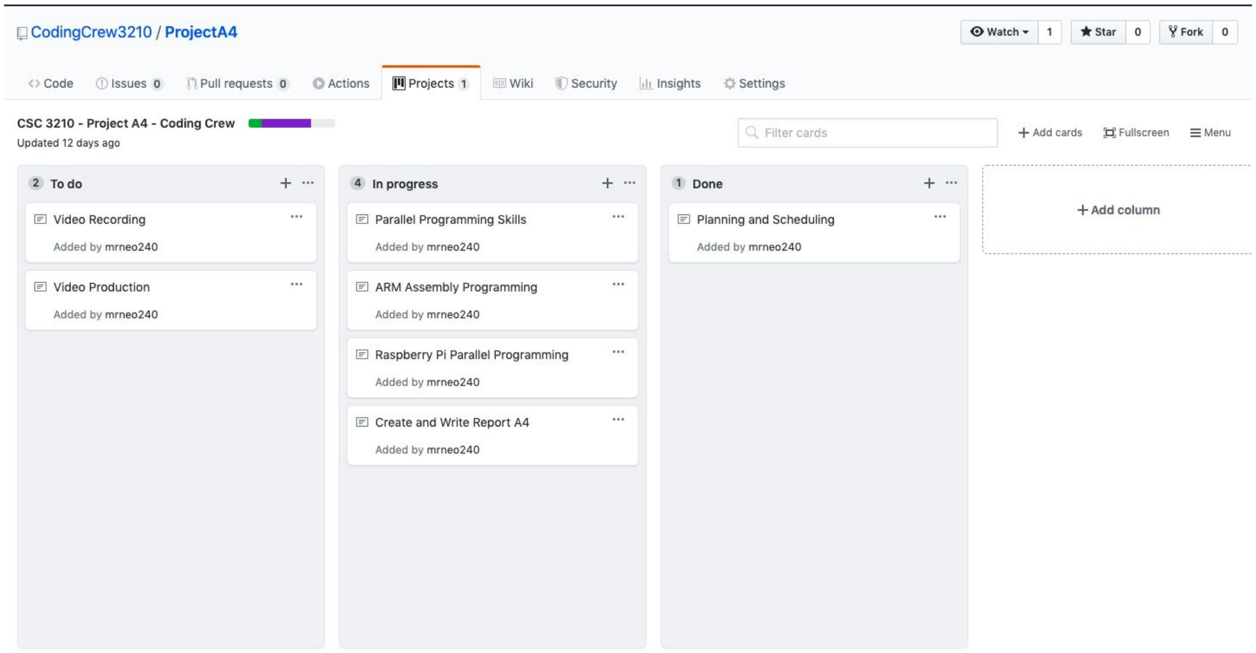
Project A4 © 2020 Coding Crew

`Hayden Kowalchuk`, `Vera Agyiri`, `Mohammad Munsur(Rohan)`, `Ping He`

Objectives

1. Use Slack, GitHub,and Word processorand create Videos to develop your soft skills—verbal and written communication, cooperation, decisions making, tasks identifications, planning, and scheduling, conflict resolution.
2. Identify and analyze the basics of parallel computing architectures and programmingusing OpenMP and C language.
3. Write ARM assembly programs.

Online Presence


CodingCrew3210 / ProjectA4

CSC 3210 - Project A4 - Coding Crew

Updated 12 days ago

To do (2)	In progress (4)	Done (1)
Video Recording — Added by mrneo240	Parallel Programming Skills — Added by mrneo240	Planning and Scheduling — Added by mrneo240
Video Production — Added by mrneo240	ARM Assembly Programming — Added by mrneo240	
	Raspberry Pi Parallel Programming — Added by mrneo240	
	Create and Write Report A4 — Added by mrneo240	

Link for Slack: https://app.slack.com/client/TNCC77K3K/GT668SUP2
Link for Github: https://github.com/CodingCrew3210/ProjectA4
Link for Video: https://www.youtube.com/watch?v=DCf_6QAApGU&feature=youtu.be