**Michael Knight**

**Project A4**

**Gitter-Geeks**

**Task 3:**

**Parallel Programming Skills**

1. **Foundation**

Race Condition

1. What is Race condition?

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

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

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

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

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

1. Summaries the Parallel Programming Patterns section I the “Introduction to Parallel Computing\_3.pdf” (two pages) in your own words.

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

1. In the section “Categorizing patterns” in the “Introduction to Parallel Computing\_3.pdt” compare the following:
   1. Collective synchronization(barrier) with Collective communication (reduction)

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

* 1. Master-worker with fork join

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

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

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

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

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

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

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

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

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

* 1. When can two statements be executed in parallel?

When each statement is Independent.

* 1. How can dependency be removed?

By Rearranging statements or eliminating statements

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

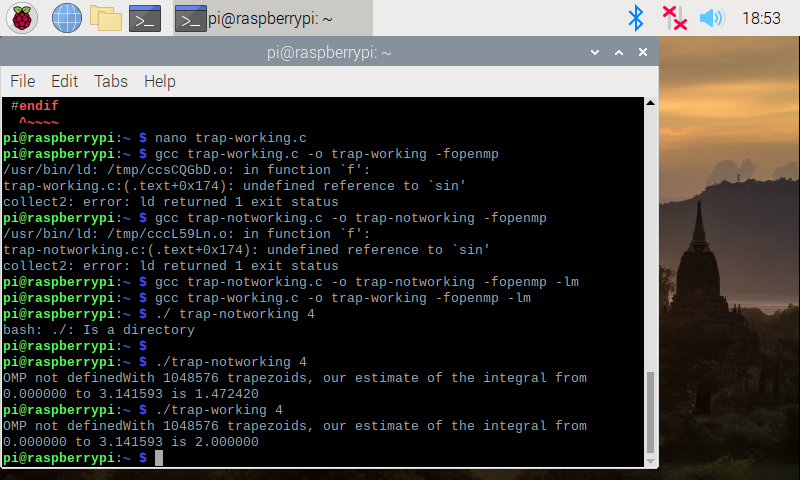
Both Statements are independent and can run parallel, the only dependent is [i] in the loop.

1. **Parallel Programing Basics**

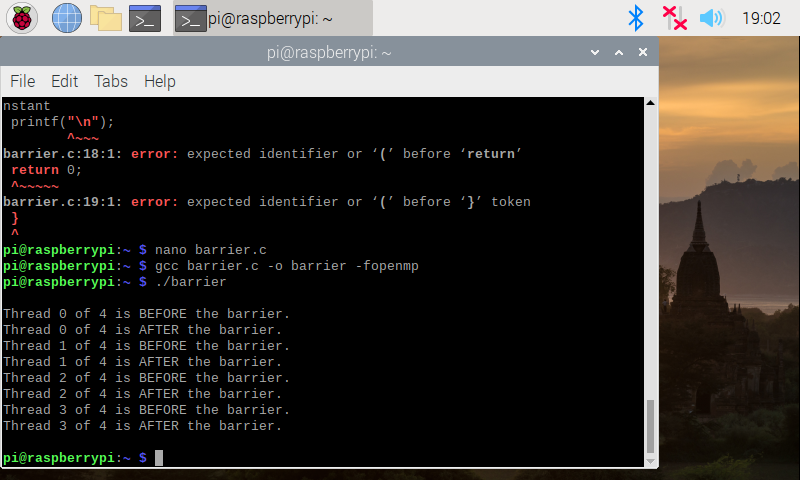
**Task 4:**

**ARM Assembly Programming**

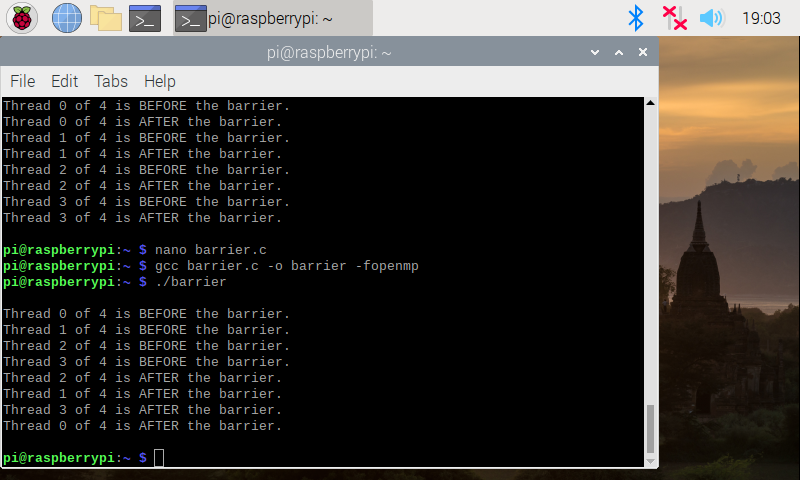
First copied and ran the program trap-notworking and trap-working to compare and observe the difference between the two.

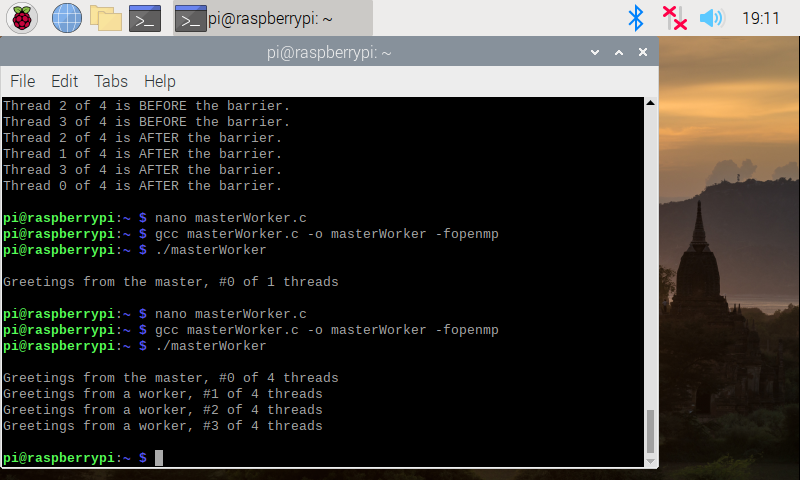


Next Copied the program Barrier and ran it.

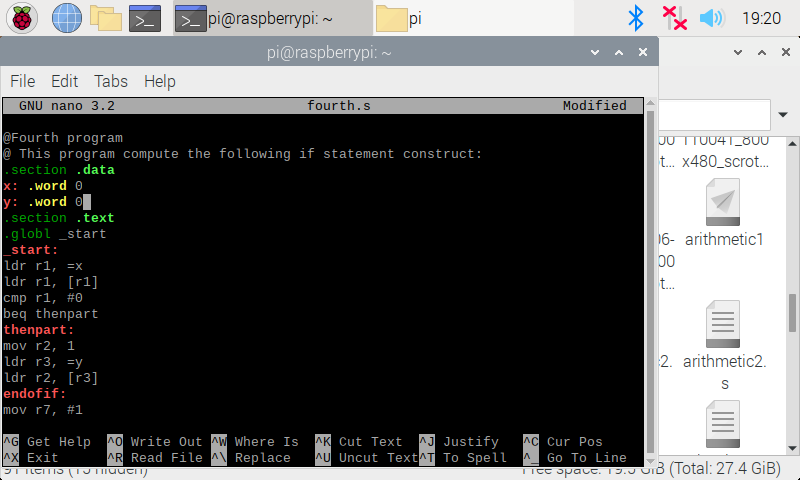


Then compared with the pragma line at 31 being uncommented out.



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

**ARM Assembly Programming:**

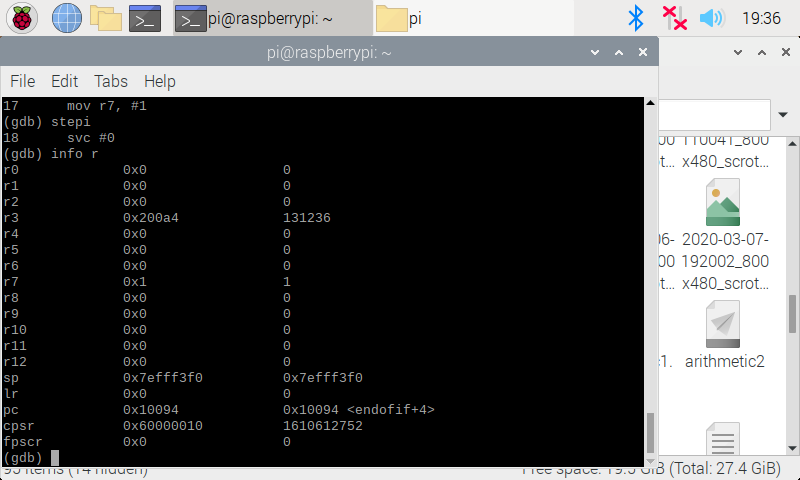
First I copied the code and saved as Fourth.s as instructed.

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

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

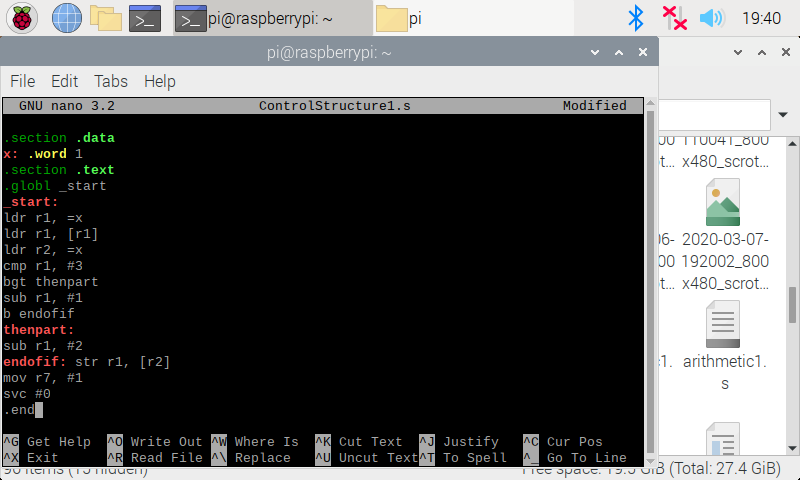
**Part 2:**

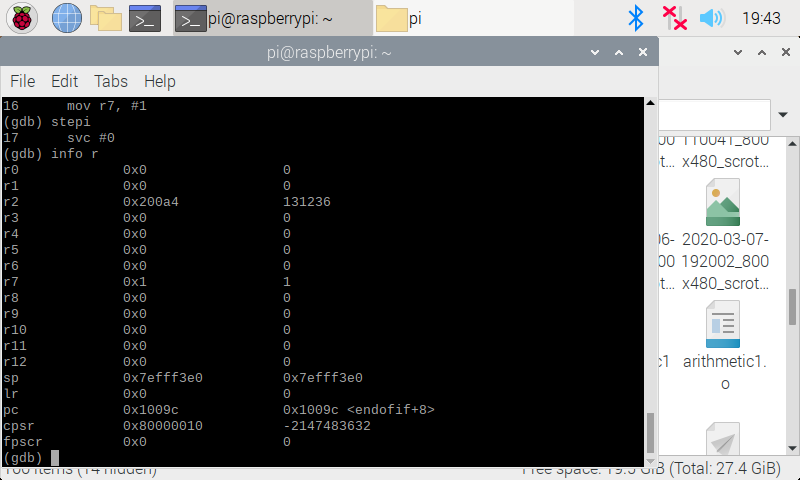
Using forth.s as a reference replaced the jump beq with bnq:



And noticed the status of the zero flag is cleared “0”

**Part 3:**

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



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