

Developing Soft and Parallel Programming Skills Using Project- Based Learning

**Spring 2019
Tuesday & Thursday 11:00 A.M. - 12:15 P.M.**

Group Name: ATLAS-SQUAD

Team Members:

Shili Guan [Team Coordinator]

Sai Rampally

Jason Poston

T'Avvion Jones

Zeak Sims

Georgia State University

Planning and Scheduling

1

Assignee Name	Email	Task	Duration (hours)	Dependency	Due Date	Note
Shili Guan [Coordinator]	sguan2@student.gsu.edu	Task 1,2,3b,4,5, and meeting at library to complete Task 6	6-7 hrs	None	3/29/19	Finish all tasks and organize the report make sure every tasks is completed before submit
T'Avvion Jones	Tjones172@student.gsu.edu	Task 3a and meeting at library to complete Task 6	3 hrs	None	3/27/19	All tasks must be finished and send to me by 03/27/19. Excellent: (100%)
Sai Rampally	srampally1@student.gsu.edu	Task 3a and meeting at library to complete Task 6 Posting on GitHub Upload Video to YouTube	3 hrs	None	3/27/19	All tasks must be finished and send to me by 03/27/19. Excellent: (100%)
Zeak Sims	zsims2@student.gsu.edu	Task 3a and meeting at library to complete Task 6	2 hrs	None	3/27/19	All tasks must be finished and send to me by 03/27/19. Excellent: (100%)
Jason Poston	jposton1@student.gsu.edu	Task 3a and meeting at library to complete Task 6	2 hrs	None	3/27/19	All tasks must be finished and send to me by 03/27/19. Excellent: (100%)

Parallel Programming Skills:

(2p) What is race condition?

The behavior of an electronics, software, or other system where the output is dependent on the sequence or timing of other uncontrollable events. These can occur in logic circuits, and in multithreaded software programs.

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

Race conditions are difficult to reproduce and debug because so much of it is caused by the timing and difference between multiple threads. The result is nondeterministic, and usually when running a debugger, the problems that occur don't show up.

(8p) How can it be fixed? Provide an example from your Project A3(see Spmd2.c)

The best way to avoid race condition is to design one's software carefully from the beginning. Being proactive and implementing an approach that will eliminate race conditions rather than reactively and try and find them using debugging software, as they might disappear. The design from Project A2, Spmd2.c shows how this can be both a good and bad implementation of it. The order in which the threads initially finish, and print happens naturally, however they're printed in such random orders and sometimes the same core does a different thread twice. Constructing a better way of tracking the IDs of each thread and which core (by initializing the ID and numThreads inside the OpenMP forking) is a better way to design this code, and helps illustrate and keep track of the fork-join program better.

15p) Summarize the Parallel Programming Patterns section in the “ Introduction to Parallel Computing_3.pdf”

Parallel programming patterns are how recurrent code are written in parallel programs. The two ways we organize these patterns are in strategies and concurrent execution mechanism. Parallel algorithmic strategies involve decisions on what exactly is to be done simultaneously by multiprocessing units. Implementation strategies focus on the structure of the algorithms as well as the structure of the entire program. Regarding concurrent execution mechanisms, they are patterns that coincide with either the system or hardware of an application and its library to enable parallelism or concurrent processing. These mechanisms have two main patterns. Process/Thread control patterns determine how the processing units are controlled during run time. Coordination Patterns arrange multiple tasks so that they run at the same time. Most software uses two major coordination patterns; message passing between processes on one multiprocessing machine or groups of computers or mutual exclusion between threads that perform procedures at the same time on a shared memory system.

Collective Synchronization (barrier) with Collective Communication (reduction)

The synchronization barrier is created so that the number of threads that are synchronizing on the barrier wait until all the other threads have reached the barrier before any other thread continues the operation. Once all the threads reach the barrier, they then execute the given task. These barriers cannot be shared across processes. The Collective Communication (reduction) threads compute their results using associative binary operator and they conclude all combining results into a global result. The reduction is an integral part of programming models such as Map Reduce where a function is applied (mapped) to all elements before they are reduced.

Master-Worker with Fork Join

In the master-worker model, the master thread divides the work to other threads known as workers, so they can gather and return back to the master thread with the acquired results. The worker processes don't know before runtime which portion of array they will handle or how many tasks they will perform. In the fork-join model, the setting up of and executing of parallel programs takes place, such that execution branches off separately in parallel at designated points in the program, so that it can merge/join at a point and resume sequential execution. Parallel sections fork recursively until a certain task is reached.

Where can we find parallelism in programming?

Parallelism can be found in programming in between statement level, between program statements, block level, loop level (can often be identified within loops) routine level, and process level. Also, parallelism can be found between larger grained statements.

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

Dependency is when one succeeding task depends on the preceding task to complete and acquire a result before other succeeding tasks are performed.

3 types of dependences: True dependences, Anti-dependences, Output dependences.

Example of True Dependency: (RAW) - read-after-write

- 1.) $A = 5$
- 2.) $B = A$
- 3.) $C = B$

Example of Anti-Dependency: (WAR) – write-after-read

- 1.) $B = 6$
- 2.) $A = B + 1$
- 3.) $B = 8$

Example of Output Dependency: (WAW) – write-after-write

- 1.) $B = 6$
- 2.) $A = B + 1$
- 3.) $B = 8$

There is an output dependency between instructions 3 and 1 so changing the ordering of instructions in this example will change the final value of A.

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

If two statements are given separate values and they are equivalent to their outcome, they are independent. If two statements are executed and their outcome was affected by the order of the statements, the two statements are dependent.

For Example:

statement1 = 1
statement2 = statement1

is dependent because the outcome of statement2 depends on the outcome of statement1 while

statement1 = 1
statement2 = 2

is independent because the order of execution of the statements does not matter.

When can two statements be executed in parallel?

Two statements can execute in parallel when there are no true dependences, anti-dependencies, output dependencies present in either statements.

How can dependency be removed?

Modifying a program by rearranging and eliminating statements can remove dependencies.

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

In order to tell if the statements are dependent, we are going to break the loops down into separate iterations and find the dependencies between the iterations. The first loop may only have one statement, but the outcome depends on what iteration the loop was on, which makes it a true dependence. The second loop, depends on which iteration it is on also making it true dependence.

Parallel Programming Basics

2.0 Integration using the trapezoidal rule

For this part, since I am the coordinator for this project, I mainly focus on this part and with the help from my team members. The part of the assignment is to observe the difference when a parallel program executes with different `#pragma omp` phrase. I just simply follow the instruction, since the codes are provided. The first program is called trap-notworking,

First, after the raspberry Pi is running, I open the Teriminal and type “nano trap-notworking.c” to open nano to copy and paste the codes, then hit “Control x” to save and exit the nano. I then use “gcc” to make executable program as “gcc trap-notworking.c -o trap-notworking -fopenmp”.

As we might know it contains bug as from the name trap-notworking. Here is the snippets of the program:

```
pi@raspberrypi:~ $ nano trap-notworking.c
pi@raspberrypi:~ $ gcc trap-notworking.c -o trap-notworking -fopenmp
/tmp/ccwrelsN.o: In function `f':
trap-notworking.c(.text+0x17c): undefined reference to `sin'
collect2: error: ld returned 1 exit status
pi@raspberrypi:~ $
```

A complete code of trap-notworking screenshot:

```

File Edit Tabs Help
GNU nano 2.7.4 File: trap-notworking.c

//The answer from this computation should be 2.0.
#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]);
}
#ifdef _OPENMP
omp_set_num_threads( threadcnt );
printf("OMP defined, threadcnt = %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) {
return sin(x);
}

```

Now, I just redo the same step to create a new program called “trap-working” as I have done for the “trap-notworking”, but I got an error by using gcc as “gcc trap-working.c -o trap-working -fopenmp”. So, I added “-lm” after -fopenmp in order to create the executable program, below is the screenshot:

```
File Edit Tabs Help
pi@raspberrypi:~ $ nano trap-working.c
pi@raspberrypi:~ $ gcc trap-working.c -o trap-working -fopenmp
/tmp/ccwRRAgX.o: In function `f':
trap-working.c:(.text+0x17c): undefined reference to `sin'
collect2: error: ld returned 1 exit status
pi@raspberrypi:~ $ gcc trap-working.c -o trap-working -fopenmp
/tmp/ccWHof3o.o: In function `f':
trap-working.c:(.text+0x17c): undefined reference to `sin'
collect2: error: ld returned 1 exit status
pi@raspberrypi:~ $ nano trap-working.c
pi@raspberrypi:~ $ gcc trap-working.c -o trap-working -fopenmp -lm
pi@raspberrypi:~ $ ./trap-working
OMP defined, threadct = 1
With 1048576 trapezoids, our estimate of the integral from
0.000000 to 3.141593 is 2.000000
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:~ $ ./trap-notworking 4
bash: ./trap-notworking: No such file or directory
pi@raspberrypi:~ $ gcc trap-notworking.c -o trap-notworking -fopenmp
/tmp/cctTHpMu.o: In function `f':
trap-notworking.c:(.text+0x17c): undefined reference to `sin'
collect2: error: ld returned 1 exit status
pi@raspberrypi:~ $
```


Then, I run the program by typing “./trap-working”, and I get the result of 2 as expect of solving the integral by the “trap-working” program. After run “./trap-working 4” which means fork 4 threads, and get the same result, but “./trap-notworking 4” didn’t work as we have done before.

Here is the complete code for “trap-working”:

```

File Edit Tabs Help
GNU nano 2.7.4 File: trap-working.c

//The answer from this computation should be 2.0.
#include <math.h>
#include <stdio.h> // printf()
#include <stdlib.h> // atoi()
#include <omp.h> // OpenMP6

/* 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]);
    }
    #ifdef _OPENMP
        omp_set_num_threads( threadcnt );
        printf("OMP defined, threadcnt = %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
 ^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Spell

By observing the output of “trap-working”, it tells that the program uses 1048576 trapezoids with our giving comment to fork 4 threads, the first thread (which is 0) is going to do the first 1048576/4 trapezoids, and so on because we didn’t explicitly write an additional static or dynamic clause to the pragma.

3.0 Coordination: Synchronization with a Barrier

For this program, we are using the concept of Barrier to create and observe a program. Frist, I open nano editor by typing “ nano barrier.c” on the terminal then after copied the code that is provided then I am using gcc to create an executable program, I repeat the steps on 2.0, only change the program name to run the program, here is the snippet of the code that **without the code #pragma omp barrier (we commented out this code)**:

```

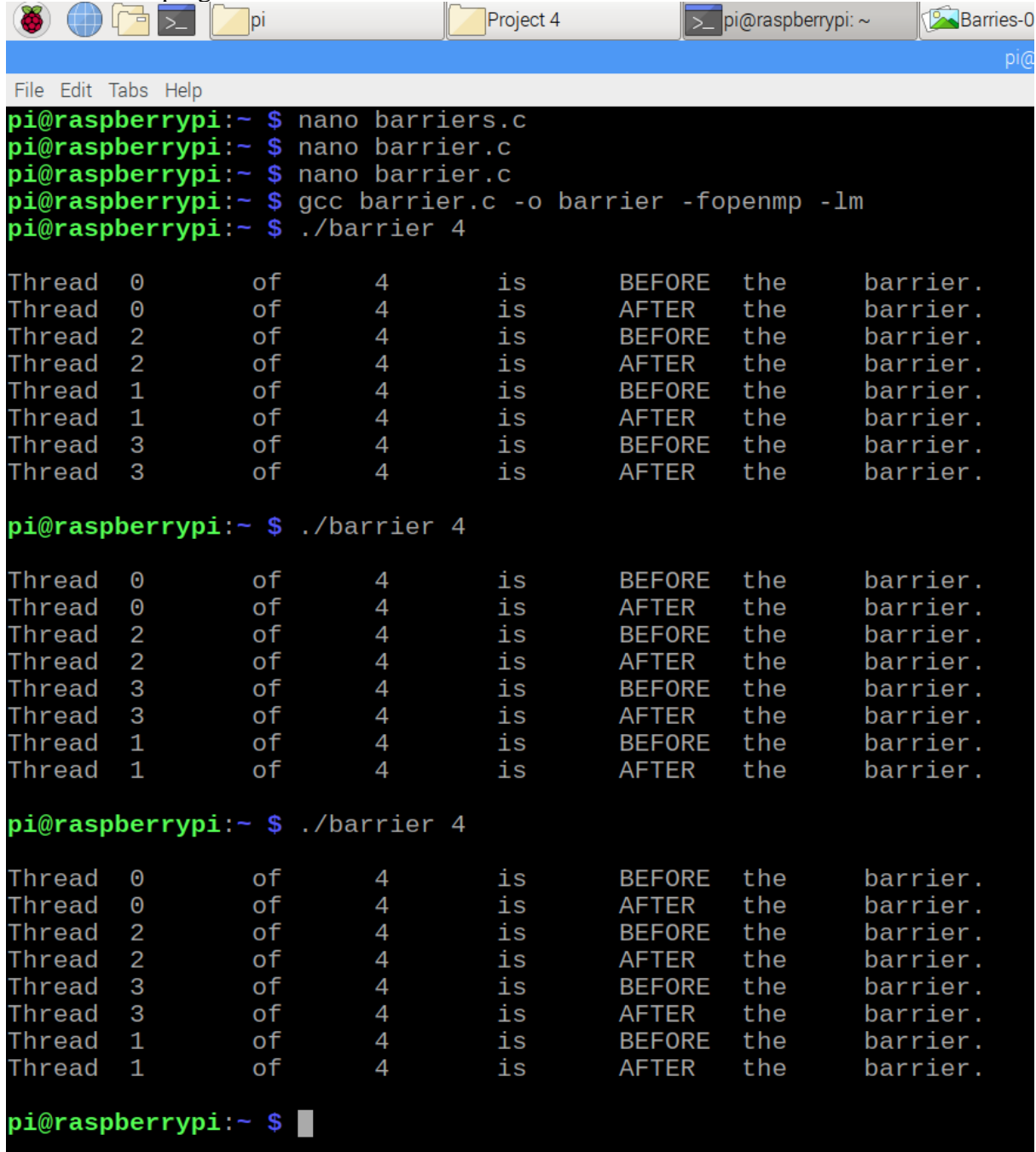
GNU nano 2.7.4                                File: barrier.c
#include <omp.h>
#include <stdlib.h>
int main(int argc, char** argv) {
    printf("\n");
    if (argc > 1) {
        omp_set_num_threads( atoi(argv[1]) );
    }
    #pragma omp parallel
    {
        int id = omp_get_thread_num();
        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 [^]Y Prev Page
[^]X Exit [^]R Read File [^]\ Replace [^]U Uncut Text [^]T To Spell [^]_ Go To Line [^]V Next Page

Then, I save and exit the nano editor. Typing `./barrier 4` will run the program and fork 4 threads of the program.



```

pi@raspberrypi:~ $ nano barriers.c
pi@raspberrypi:~ $ nano barrier.c
pi@raspberrypi:~ $ nano barrier.c
pi@raspberrypi:~ $ gcc barrier.c -o barrier -fopenmp -lm
pi@raspberrypi:~ $ ./barrier 4

Thread 0      of      4      is      BEFORE the      barrier.
Thread 0      of      4      is      AFTER  the      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.

pi@raspberrypi:~ $ ./barrier 4

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

pi@raspberrypi:~ $ ./barrier 4

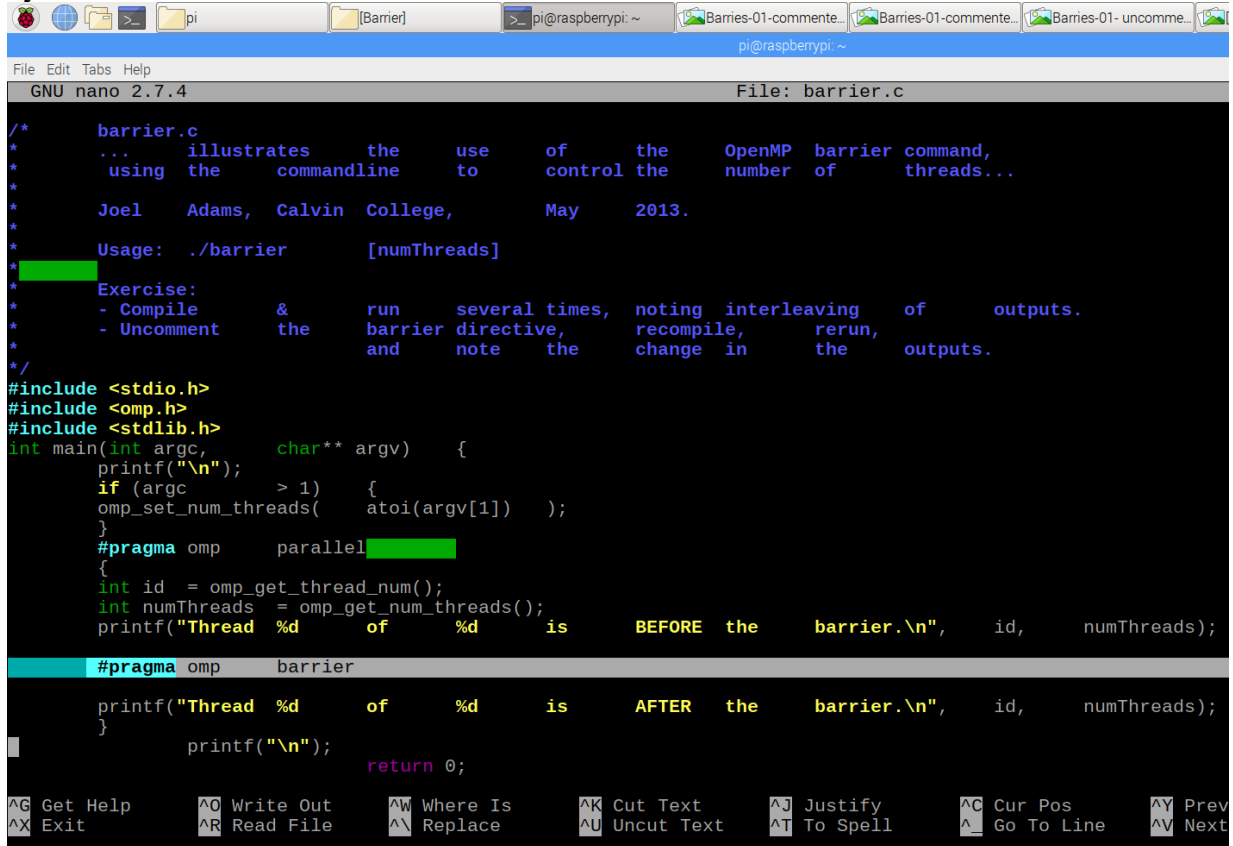
Thread 0      of      4      is      BEFORE the      barrier.
Thread 0      of      4      is      AFTER  the      barrier.
Thread 2      of      4      is      BEFORE the      barrier.
Thread 2      of      4      is      AFTER  the      barrier.
Thread 3      of      4      is      BEFORE the     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

pi@raspberrypi:~ $ █

```

I run the program few times to observe the output, as the snippet shows above that the program executes the same thread of “BEFORE” and “AFTER” before it executes next thread.

Next, I need to exam what the output looks like with the code **#pragma omp barrier** by uncommenting it.

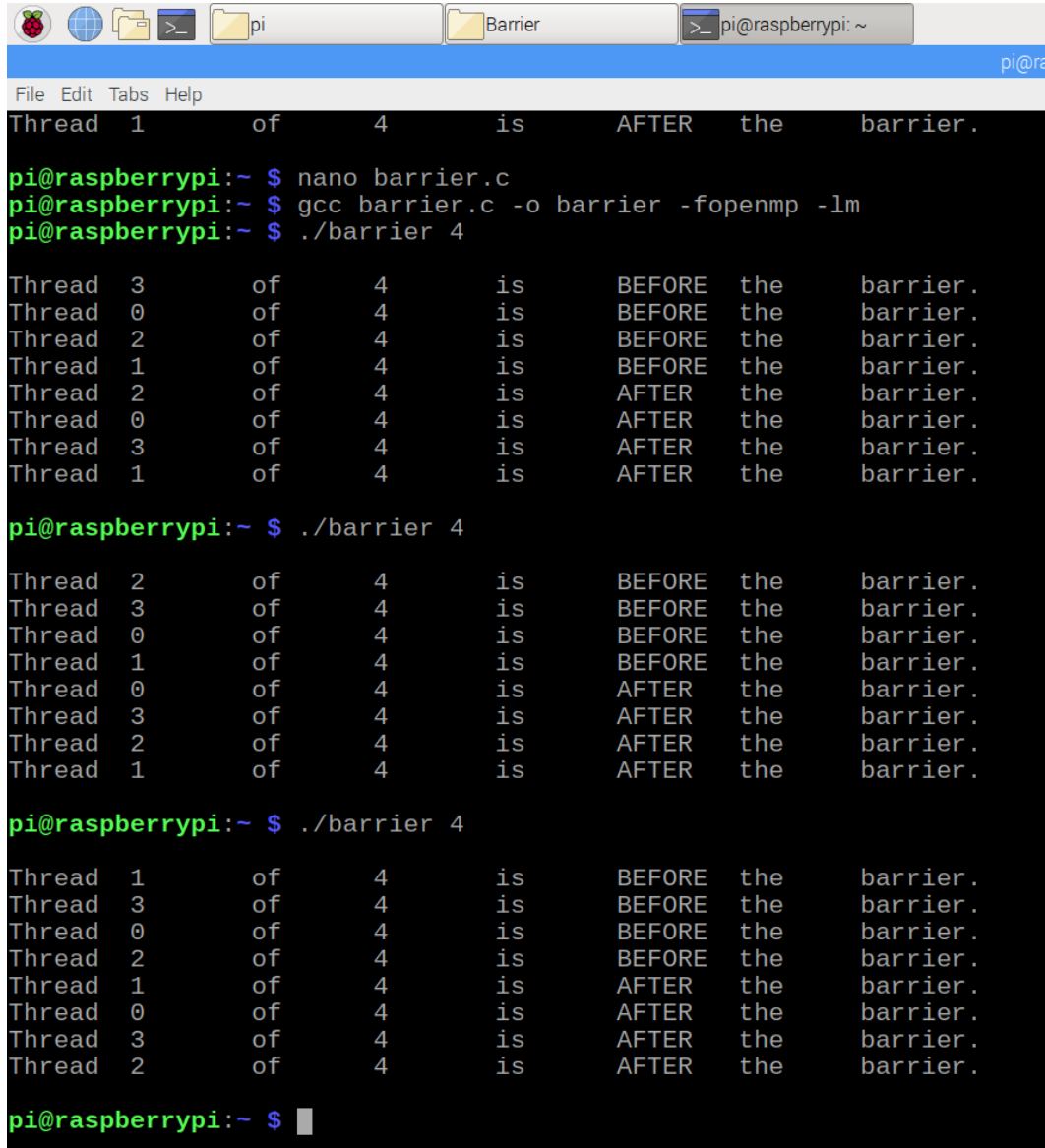


```

/*
 * 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;
}

```

After I recompiled, I write `./barrier 4` on the terminal again, with the new code in, I got a new results as we might be able to guess, I also run few times to get some ideas about how it works



```

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

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

pi@raspberrypi:~ $ ./barrier 4

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

pi@raspberrypi:~ $ ./barrier 4

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

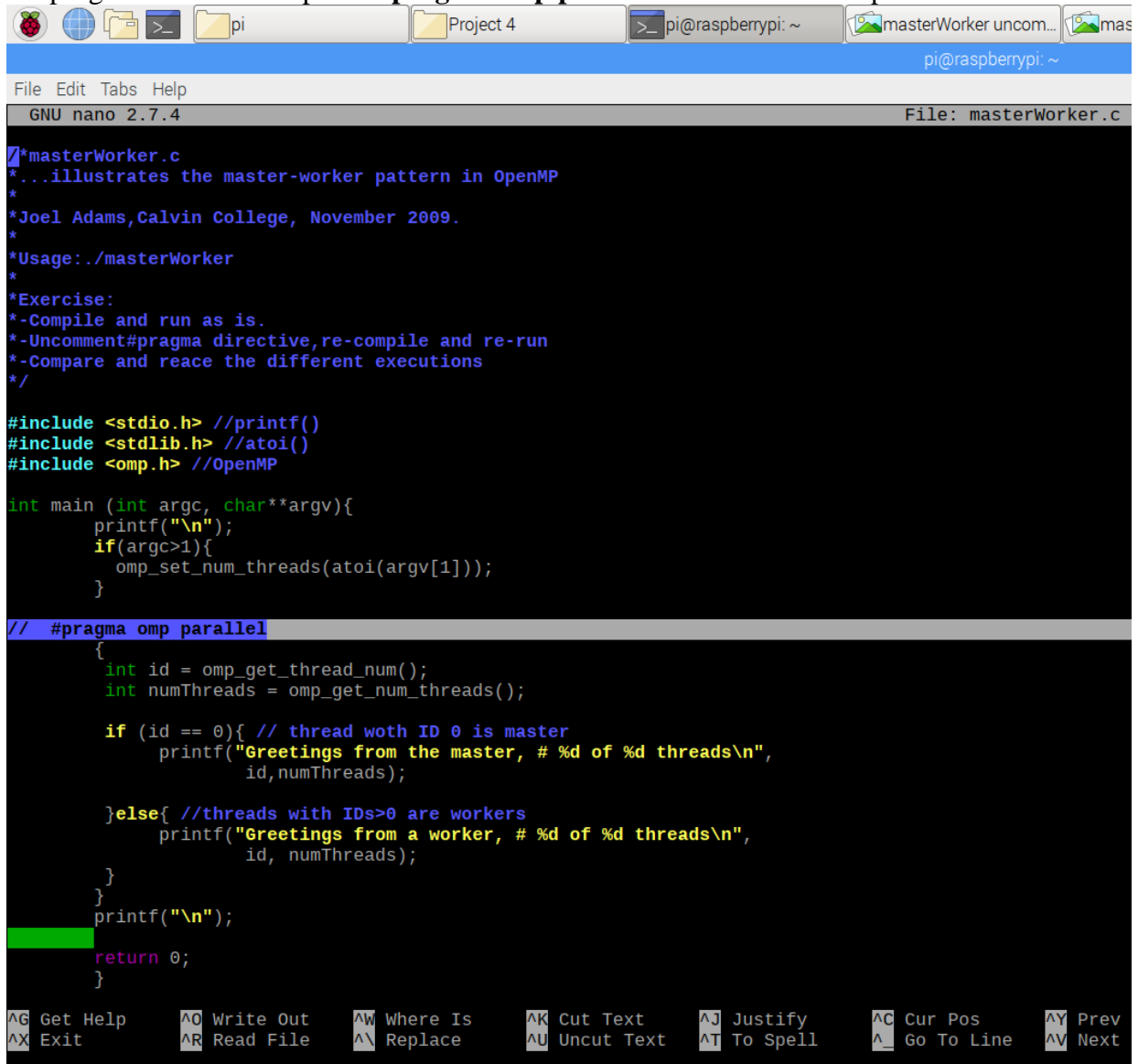
pi@raspberrypi:~ $

```

It finished executing 4 threads BEFORE, and then continues to execute the AFTER as we can see from the output. This is because it contains barrier pattern. When barrier is used in a parallel program, it makes sure that all threads finish a parallel section of code before moving on to the next section.

4.0 Program Structure: The Master-Worker Implementation Strategy

Once I open the nano editor and copy the code in and save to create a program called “masterWorker” I use gcc to compile the program and run it with `./masterWorker 4`. First, I run the program **without** the phase `#pragma omp parallel` to observe the output



```

/*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 reace the different executions
*/

#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");
return 0;
}

```

Then, exit, save, compile, and then run the program few times

```

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

Greetings from the master, # 0 of 1 threads

pi@raspberrypi:~ $ ./masterWorker 4

Greetings from the master, # 0 of 1 threads

pi@raspberrypi:~ $ ./masterWorker 4

Greetings from the master, # 0 of 1 threads

pi@raspberrypi:~ $ █

```

From the output, we can see that only one thread is called master is being execute. Then, I add the phase **#pragma omp parallel** by uncommented it to see maybe we can get something new.

```

File Edit Tabs Help
GNU nano 2.7.4 File: masterWorker.c

/*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 reace the different executions
*/

#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]));
    }

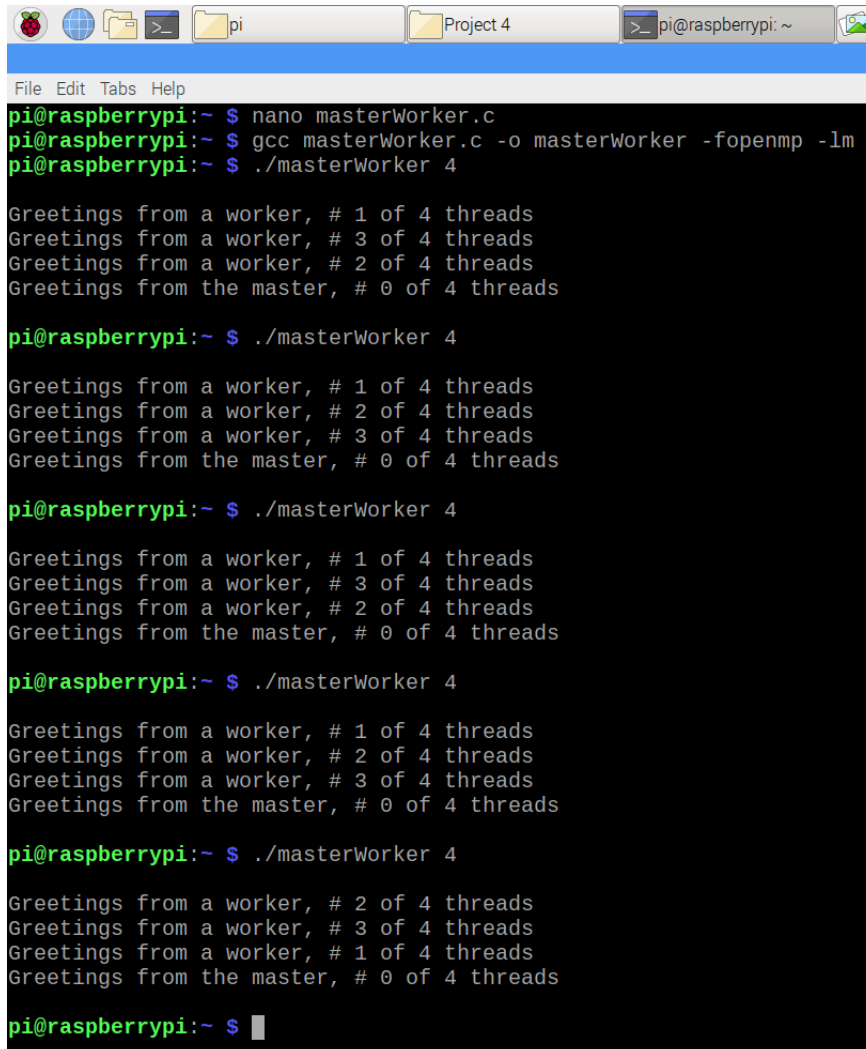
    #pragma omp parallel
    {
        int id = omp_get_thread_num();
        int numThreads = omp_get_num_threads();

        if (id == 0){ // thread woth 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 ^Y Prev P
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Spell ^_ Go To Line ^V Next P

```

After running the new program few times, I get three more threads called worker on the output, as the output shows that the worker threads are in random order, but execute before the master thread's block of code.



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

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:~ $ ./masterWorker 4

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

pi@raspberrypi:~ $ ./masterWorker 4

Greetings from a worker, # 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:~ $ ./masterWorker 4

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

pi@raspberrypi:~ $ ./masterWorker 4

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

pi@raspberrypi:~ $
```

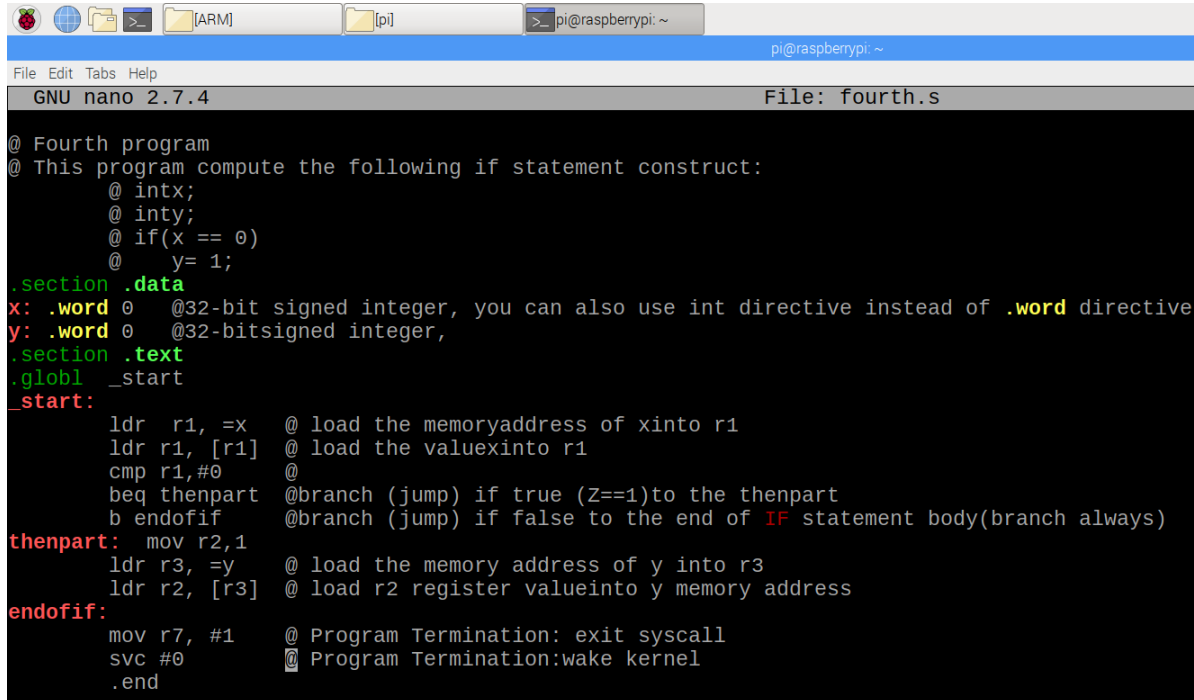
This is because the program contains one thread that execute one block of code when it forks, it is called master, and the other three thread are called workers which execute in different block of code when they fork.

ARM Assembly Programming

Part 1: fourth program

This part is very simple. I need to copy the code from handout and then use command “**as -g -o fourth.o fourth.s**” to assemble it first then, link it with “**ld -o fourth fourth.o**”, after I linked the program, I need to use gdb to debug it, type “**gdb fourth**”.

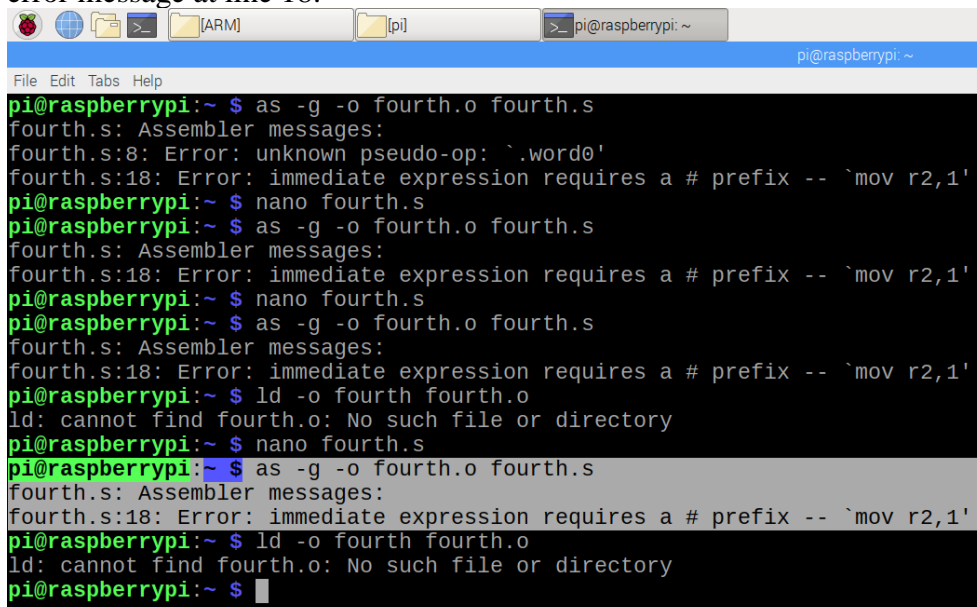
Given code:



```

@ 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 memoryaddress of xinto r1
    ldr r1, [r1] @ load the valuexinto 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 valueinto y memory address
endofif:
    mov r7, #1 @ Program Termination: exit syscall
    svc #0 @ Program Termination:wake kernel
    .end
  
```

After save and exit, I use the command to assemble and link, but when I debug the code, I got an error message at line 18:

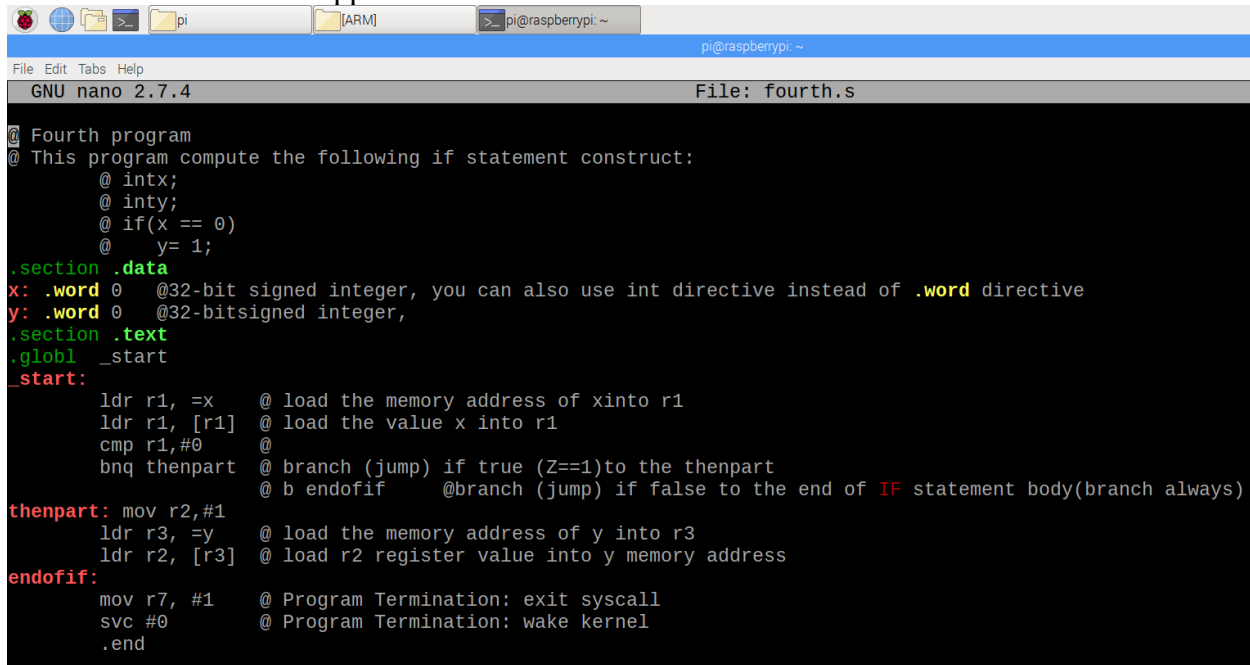


```

pi@raspberrypi:~$ as -g -o fourth.o fourth.s
fourth.s: Assembler messages:
fourth.s:8: Error: unknown pseudo-op: `.word0'
fourth.s:18: Error: immediate expression requires a # prefix -- `mov r2,1'
pi@raspberrypi:~$ nano fourth.s
pi@raspberrypi:~$ as -g -o fourth.o fourth.s
fourth.s: Assembler messages:
fourth.s:18: Error: immediate expression requires a # prefix -- `mov r2,1'
pi@raspberrypi:~$ nano fourth.s
pi@raspberrypi:~$ as -g -o fourth.o fourth.s
fourth.s: Assembler messages:
fourth.s:18: Error: immediate expression requires a # prefix -- `mov r2,1'
pi@raspberrypi:~$ ld -o fourth fourth.o
ld: cannot find fourth.o: No such file or directory
pi@raspberrypi:~$ nano fourth.s
pi@raspberrypi:~$ as -g -o fourth.o fourth.s
fourth.s: Assembler messages:
fourth.s:18: Error: immediate expression requires a # prefix -- `mov r2,1'
pi@raspberrypi:~$ ld -o fourth fourth.o
ld: cannot find fourth.o: No such file or directory
pi@raspberrypi:~$
  
```

because I try to move immediate to memory so I need to fix it by rewrite **mov r2, #1**

Here is the fixed code snippet:

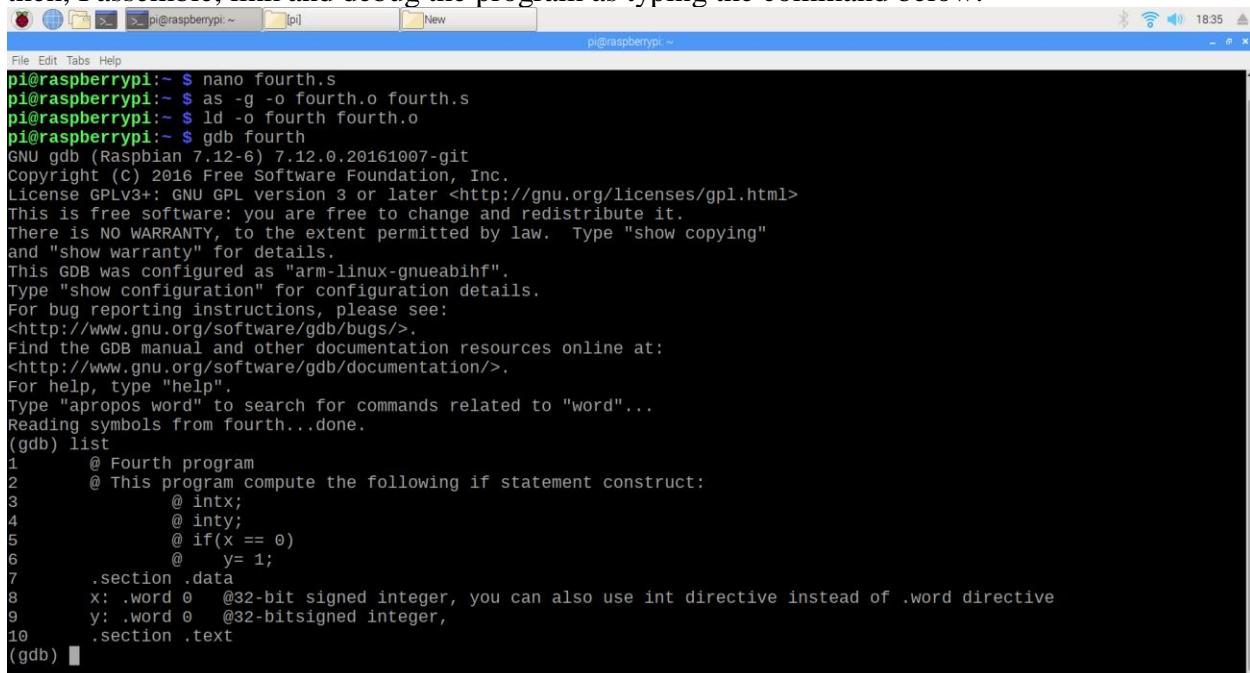


```

GNU nano 2.7.4 File: fourth.s
@ 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-bitsigned 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 @
    bneq 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

```

then, I assemble, link and debug the program as typing the command below:



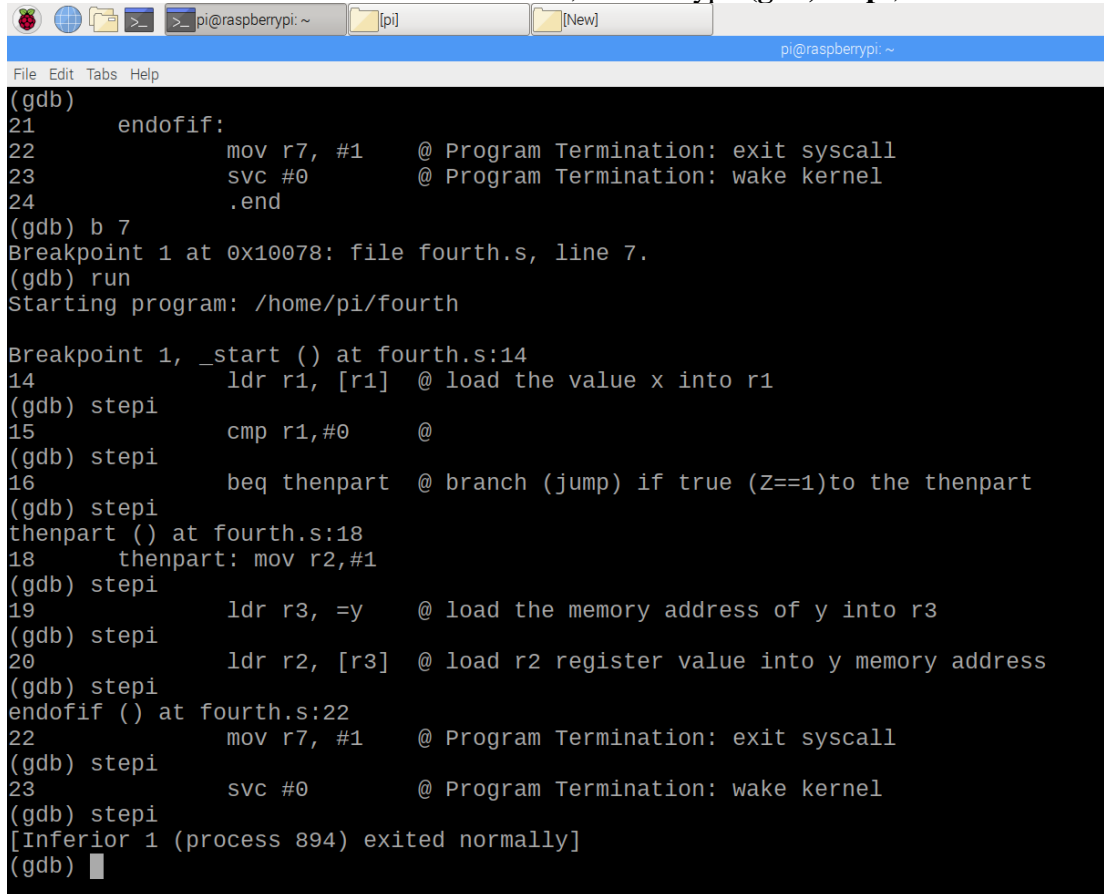
```

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 7.12-6) 7.12.0.20161007-git
Copyright (C) 2016 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-gnueabi".
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      @ 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 directive
9      y: .word 0 @32-bitsigned integer,
10     .section .text
(gdb)

```

I also use command **list** to list the first 10 lines of the code, if I want to keep showing the next 10 lines, I just hit the enter key.

In order to run the program, I need to set a breakpoint. **(gdb) b 7** will set my break point at line 7, then typing **(gdb) run**. This will allow the debugger to run automatically until it stops. If we want to continue to the next execution, we can type **(gdb) stepi**, until it exits the program.



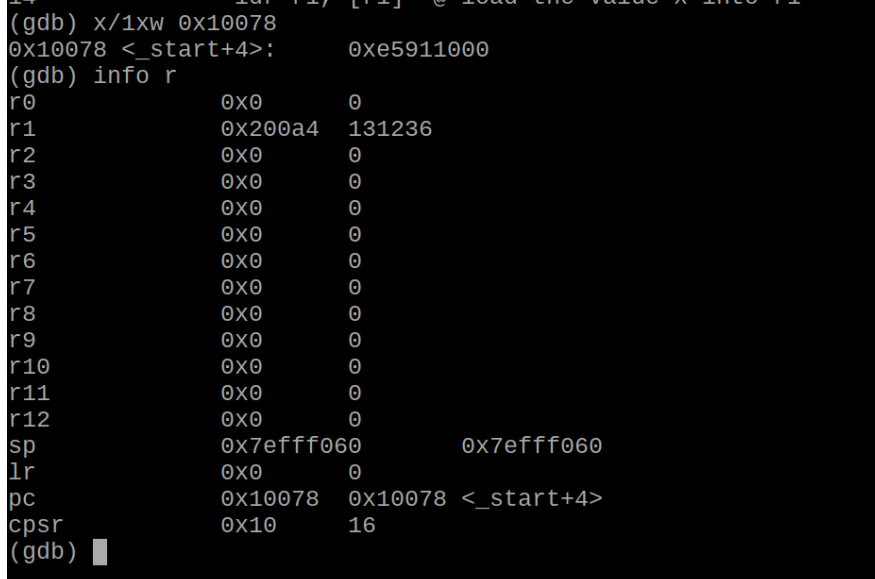
```

pi@raspberrypi: ~
File Edit Tabs Help
(gdb)
21     endif:
22         mov r7, #1    @ Program Termination: exit syscall
23         svc #0        @ Program Termination: wake kernel
24         .end
(gdb) b 7
Breakpoint 1 at 0x10078: file fourth.s, line 7.
(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 thenpart
(gdb) stepi
thenpart () at fourth.s:18
18     thenpart: mov r2, #1
(gdb) stepi
19         ldr r3, =y    @ load the memory address of y into r3
(gdb) stepi
20         ldr r2, [r3]  @ load r2 register value into y memory address
(gdb) stepi
endif () at fourth.s:22
22         mov r7, #1    @ Program Termination: exit syscall
(gdb) stepi
23         svc #0        @ Program Termination: wake kernel
(gdb) stepi
[Inferior 1 (process 894) exited normally]
(gdb)

```

We can ask for certain information from the register by enter **x/1xw** (the number shows in your gdb). In our case is 0x10078, so I type **x/1xw 0x10078**. I got **0xe5911000** where our 0x10078 in hex are in our memory. The **Z-flag** value is **0** which is unset.



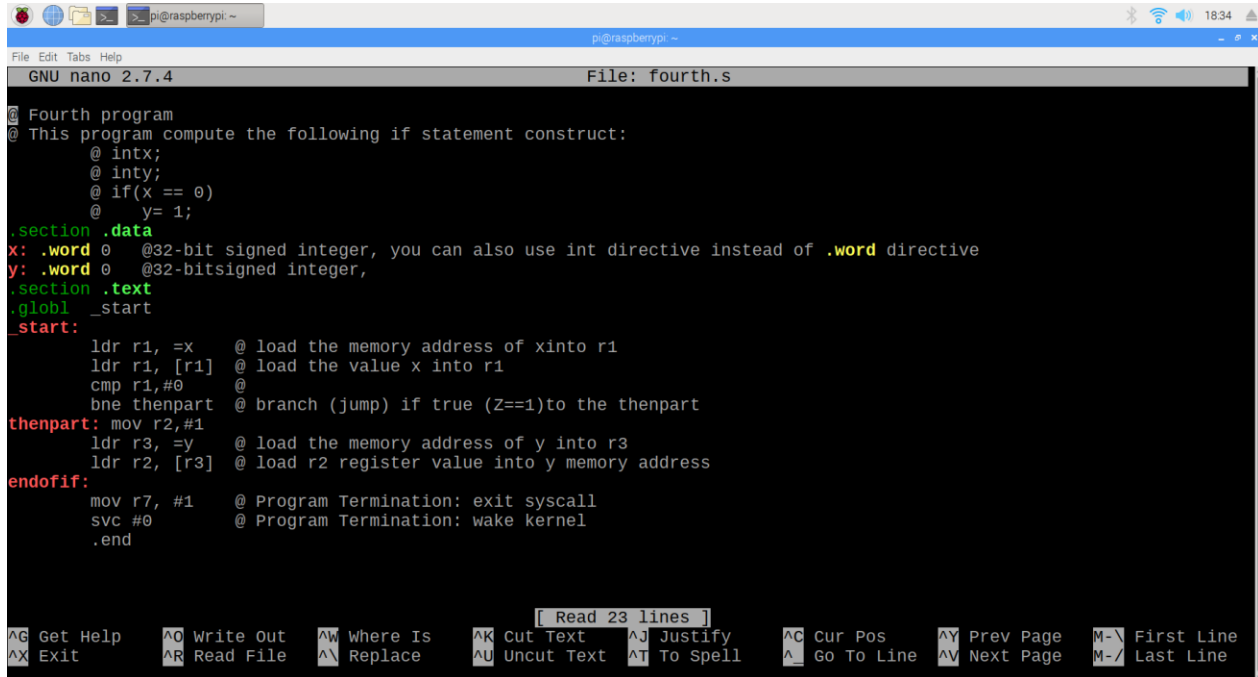
```

(gdb) x/1xw 0x10078
0x10078 <_start+4>:      0xe5911000
(gdb) info r
r0                0x0      0
r1                0x200a4  131236
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                0x7efff060 0x7efff060
lr                0x0      0
pc                0x10078 0x10078 <_start+4>
cpsr              0x10     16
(gdb)

```

Part 2:

We are using the same code from part 1, but replace the **beq** to **bne**, then remove the instruction **b** from code.

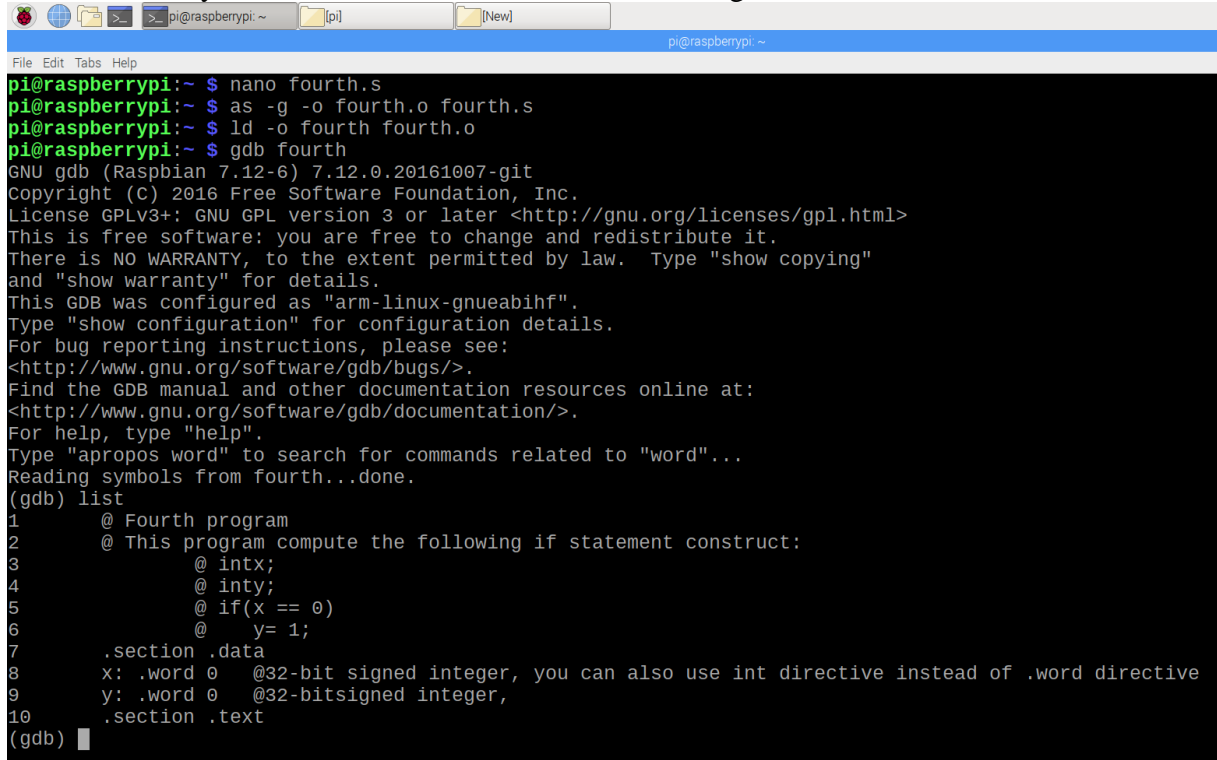


```

GNU nano 2.7.4 File: fourth.s
@ 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-bitsigned 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
endif:
    mov r7, #1 @ Program Termination: exit syscall
    svc #0 @ Program Termination: wake kernel
.end

```

Then repeat part 1: assemble, link, run and debug, then observe the output and where our value store in memory address, also what is the value for Z-flag.



```

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 7.12-6) 7.12.0.20161007-git
Copyright (C) 2016 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-gnueabi".
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      @ 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 directive
9      y: .word 0 @32-bitsigned integer,
10     .section .text
(gdb)

```

Again, we set the breakpoint first, and then run the program and get the register information by command **x/1xw (the number shows in your gdb)**. I still have the same value as part 1. 0x10078 store at the memory address **0xe5911000** and the Z-flag values is the same as 0 unset.

Part 2 code just an improve version of part 1, it is more efficient in time than part 1 because the code **b** instruction means a back-to-back branches. In part 2, we remove that instruction to avoid this problem.

Part 3: Create a program called “ControlStructure1”

In this part, I need to create my own program that run the IF statement.

```
If x <= 3
    x = x - 1
else
    x = x - 2
```

Since I have already get used to ARM programming, and knowing how to assemble, link, debug and run the program. Also that we just learn IF statement in X86 within weeks so the memory still fresh.

Here, is the code:

I set the initial value of **x = 0**

```
File Edit Tabs Help
GNU nano 2.7.4 File: ControlStructure1.s
pi@raspberrypi: ~
[Project 4]
[2019-03-26-131147...]
pi@raspberrypi: ~

@This is programming part 3 of project4
@if x <= 3
@   x = x - 1
@else
@   x = x - 2
.section .data
x: .word 0

.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
    ldr r2,=x        @load the memory address of x into r2

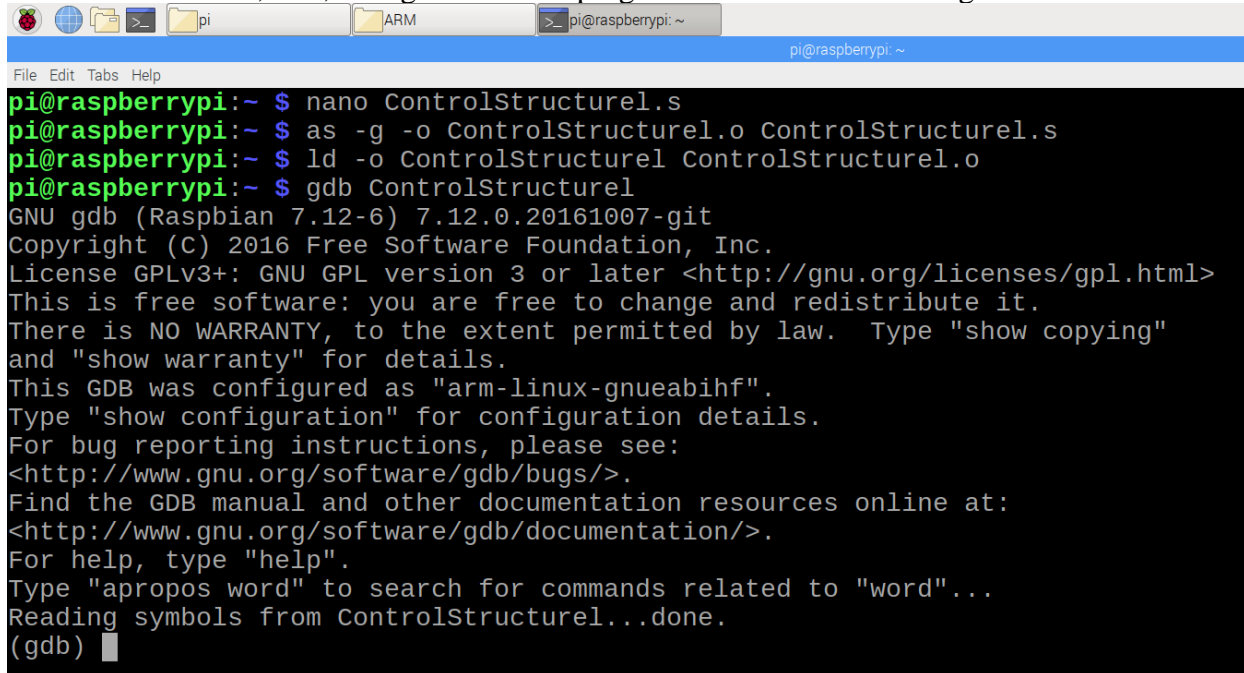
    cmp r1,#3        @
    ble thenpart     @branch(jump) if true(x<=3) to the thenpart
    sub r1,#2        @subtract 2 from r1
    b endofif        @branch (jump) of false to the end of IF statement body(branch always)

thenpart:
    sub r1,#1        @subtract 1 from r1

endofif:
    str r1,[r2]       @store r2 value to r1
    mov r7,#1        @program termination: exit syscall

^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify     ^C Cur Pos     ^Y Pr
^X Exit        ^R Read File   ^\ Replace     ^U Uncut Text  ^T To Spell    ^_ Go To Line   ^V Ne
```

Next is to assemble, link, debug and run the program. As I do the following:

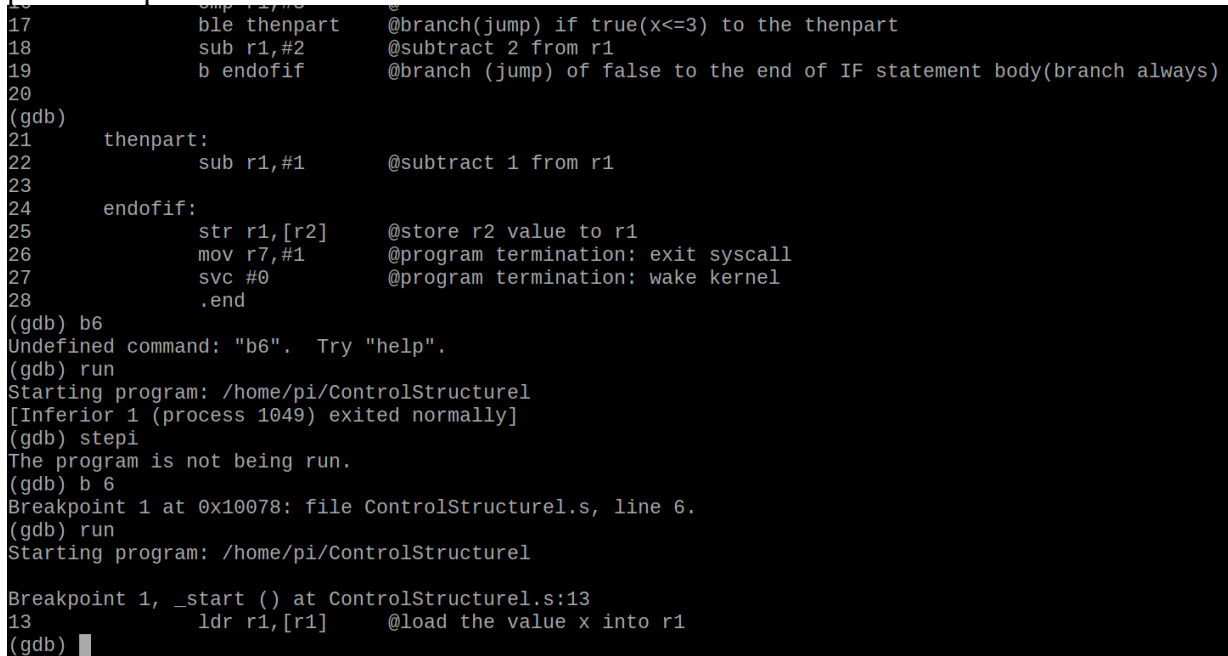


```

pi@raspberrypi:~ $ nano ControlStructure1.s
pi@raspberrypi:~ $ as -g -o ControlStructure1.o ControlStructure1.s
pi@raspberrypi:~ $ ld -o ControlStructure1 ControlStructure1.o
pi@raspberrypi:~ $ gdb ControlStructure1
GNU gdb (Raspbian 7.12-6) 7.12.0.20161007-git
Copyright (C) 2016 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-gnueabi".
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 ControlStructure1...done.
(gdb)

```

Setting break point at line 6, then run and getting information from register by doing the steps as part 1 and part 2:



```

17      ble thenpart      @branch(jump) if true(x<=3) to the thenpart
18      sub r1,#2          @subtract 2 from r1
19      b endifif          @branch (jump) of false to the end of IF statement body(branch always)
20
(gdb)
21      thenpart:
22          sub r1,#1      @subtract 1 from r1
23
24      endifif:
25          str r1,[r2]     @store r2 value to r1
26          mov r7,#1       @program termination: exit syscall
27          svc #0          @program termination: wake kernel
28          .end
(gdb) b6
Undefined command: "b6".  Try "help".
(gdb) run
Starting program: /home/pi/ControlStructure1
[Inferior 1 (process 1049) exited normally]
(gdb) stepi
The program is not being run.
(gdb) b 6
Breakpoint 1 at 0x10078: file ControlStructure1.s, line 6.
(gdb) run
Starting program: /home/pi/ControlStructure1

Breakpoint 1, _start () at ControlStructure1.s:13
13      ldr r1,[r1]         @load the value x into r1
(gdb)

```

I also have used **stepi** to exam my program one step at a time to observe if the program does as I want it to:

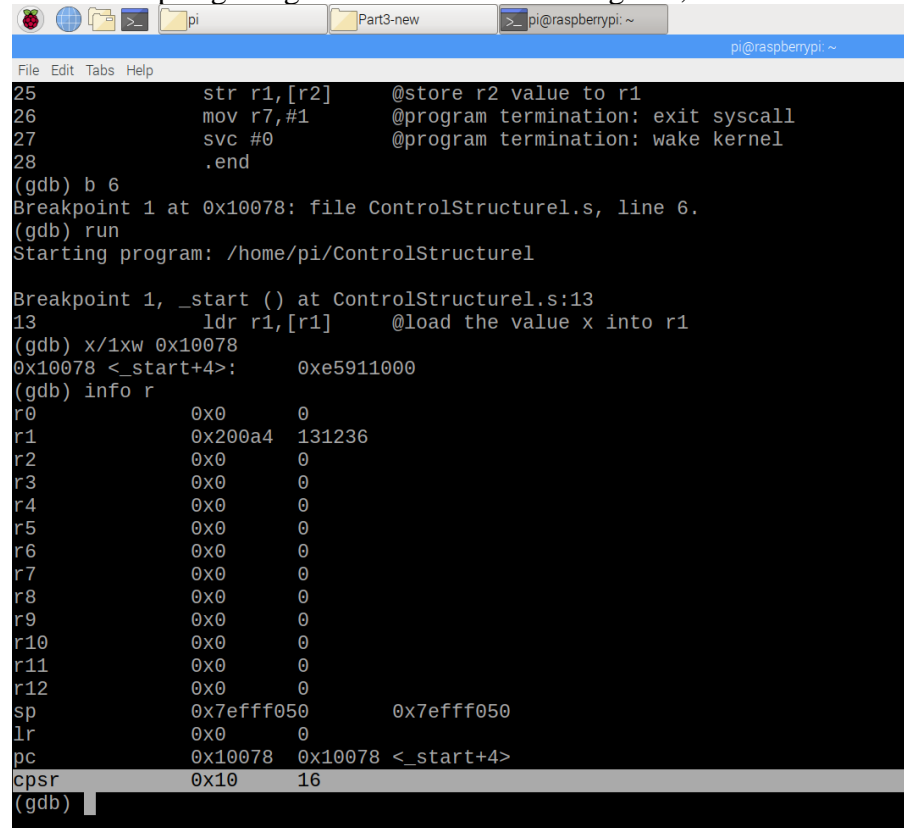
```

Breakpoint 1 at 0x10078: file ControlStructure1.s, line 6
(gdb) run
Starting program: /home/pi/ControlStructure1

Breakpoint 1, _start () at ControlStructure1.s:13
13      ldr r1,[r1]      @load the value x into r1
(gdb) stepi
14      ldr r2,=x        @load the memory address of x into r2
(gdb) stepi
16      cmp r1,#3        @
(gdb) stepi
17      ble thenpart     @branch(jump) if true(x<=3) to the thenpart
(gdb) stepi
thenpart () at ControlStructure1.s:22
22      sub r1,#1        @subtract 1 from r1
(gdb) stepi
endif () at ControlStructure1.s:25
25      str r1,[r2]      @store r2 value to r1
(gdb) stepi
26      mov r7,#1        @program termination: exit syscall
(gdb) stepi
27      svc #0           @program termination: wake kernel
(gdb) stepi
[Inferior 1 (process 1083) exited normally]
(gdb) █

```

The last step is getting information from the register, and what is the values for Z-flag



```

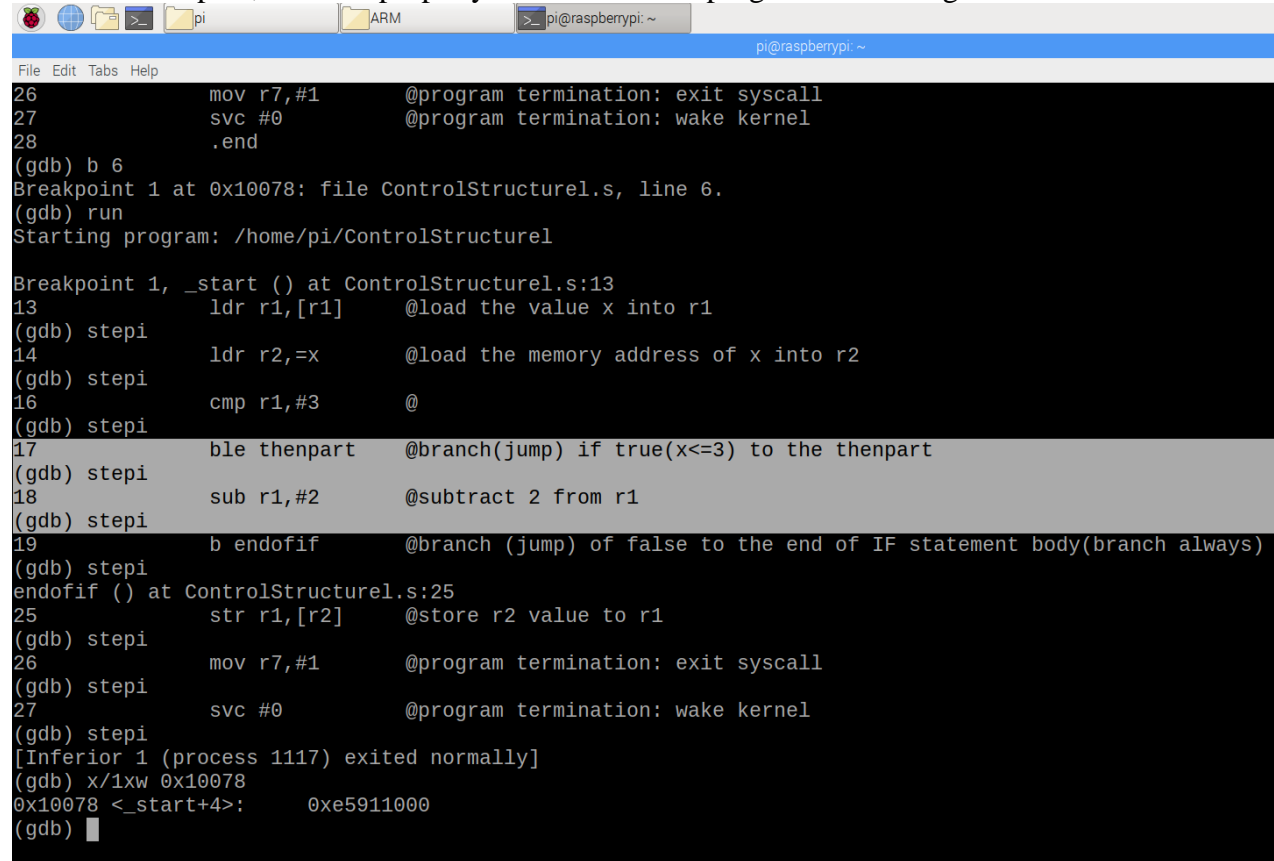
File Edit Tabs Help
25      str r1,[r2]      @store r2 value to r1
26      mov r7,#1        @program termination: exit syscall
27      svc #0           @program termination: wake kernel
28      .end
(gdb) b 6
Breakpoint 1 at 0x10078: file ControlStructure1.s, line 6.
(gdb) run
Starting program: /home/pi/ControlStructure1

Breakpoint 1, _start () at ControlStructure1.s:13
13      ldr r1,[r1]      @load the value x into r1
(gdb) x/1xw 0x10078
0x10078 <_start+4>:      0xe5911000
(gdb) info r
r0          0x0         0
r1          0x200a4     131236
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          0x7efff050   0x7efff050
lr          0x0         0
pc          0x10078     0x10078 <_start+4>
cpsr       0x10        16
(gdb) █

```


Oddly enough, I do get the same value as part 1 and part 2 that the value is store at memory address 0xe591000 and the Z-flag values is 0, still unset.

I also set **x = 6** to exam my program and be sure it actually works, by using **stepi** to execute the program one step at a time (see snippet below), It execute the line **sub r1,#2** which is subtract 2 to the x. As expect, it works properly to show that the program is working.



```

File Edit Tabs Help
26      mov r7,#1      @program termination: exit syscall
27      svc #0        @program termination: wake kernel
28      .end
(gdb) b 6
Breakpoint 1 at 0x10078: file ControlStructure1.s, line 6.
(gdb) run
Starting program: /home/pi/ControlStructure1

Breakpoint 1, _start () at ControlStructure1.s:13
13      ldr r1,[r1]     @load the value x into r1
(gdb) stepi
14      ldr r2,=x       @load the memory address of x into r2
(gdb) stepi
16      cmp r1,#3      @
(gdb) stepi
17      ble thenpart   @branch(jump) if true(x<=3) to the thenpart
(gdb) stepi
18      sub r1,#2      @subtract 2 from r1
(gdb) stepi
19      b endofif      @branch (jump) of false to the end of IF statement body(branch always)
(gdb) stepi
endofif () at ControlStructure1.s:25
25      str r1,[r2]    @store r2 value to r1
(gdb) stepi
26      mov r7,#1      @program termination: exit syscall
(gdb) stepi
27      svc #0        @program termination: wake kernel
(gdb) stepi
[Inferior 1 (process 1117) exited normally]
(gdb) x/1xw 0x10078
0x10078 <_start+4>: 0xe5911000
(gdb) █

```


Appendix

Slack Account: <https://atlas-squad.slack.com/messages/CFR6D9LHJ/team/>

GitHub: <https://github.com/ATLAS-SQUAD/CSc-3210>

Youtube Channel: <https://www.youtube.com/watch?v=zx91HARFPto#action=share>

