

# ***Developing Soft and Parallel Programming Skills Using Project - Based Learning***

*Spring - 2019*

*Awad Winning Group*

*Donald Weaver, Kishan Bhakta, Mindy Tran,  
Sean Staley, Shilp Patel*

### **Planning and Scheduling**

For the fourth project we are more comfortable with each other and are more aware of each others capabilities. We quickly assigned tasks and had plenty of time to complete said tasks thanks to the long break. At this point everyone has gotten to use the pi so those that are really interested got to double dip. We were also still able to find good times to meet up to discuss the planning, as well as the questions and progress. And of course slack has continued to be a great platform to coordinate and communicate.

<b>Name</b>	<b>Email</b>	<b>Task</b>	<b>Duration (hours)</b>	<b>Dependency</b>	<b>Due Date</b>	<b>Note</b>
Donald Weaver(coordinator)	dweaver20@student.gsu.edu	Report	3	report, table	3/29/19	print the report
Sean Staley	sstaley4@student.gsu.edu	Parallel Programming Basics	3	Parallel programming lab and report	3/28/19	Take screenshots
Shilp Patel	Spatel255@student.gsu.edu	Video and Github	2	recording and editing video, GitHub	3/29/19	Make sure to include Sean's video
Mindy Tran	mtran42@student.gsu.edu	Parallel Programming Skills	1	Led and documented our discussion	3/27/19	Make sure everyone understands the questions
Kishan Bhakta	kbhakta1@student.gsu.edu	ARM assembly program	5	ARM assembly program lab and report	3/28/19	Take screenshots

## **Parallel Programming Skills**

### *1. What is race condition?*

A race condition/race hazard is the behavior of an electronics, software, or other systems where the output is dependent on the sequence or timing of other uncontrollable events.

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

It is difficult to reproduce and debug because it's a Heisenbug. A Heisenbug is a software bug that disappears when running in debug mode, when additional logging is added, or when attaching a debugger. This means the result is nondeterministic and depends on the relative timing between interfering threads.

### *3. How can it be fixed? Provide an example from your Project\_A3 (see spmd2.c)*

It is better to avoid race conditions by careful software design rather than attempting to fix them afterward because it's a Heisenbug. For example, in Project\_A2, spmd2.c was not designed carefully because the threads were sharing memory location of the same variable. They did not have their own copy of the variable, so you must fix it by make full variable declaration in the fork-thread.

### *4. Summarizes the Parallel Programming Patterns section in the "Introduction to Parallel Computing\_3.pdf" (two pages) in your own words (one paragraph, no more than 150 words.)*

Parallel patterns contain many patterns that have been developed over time so newcomers can learn the most efficient pattern. The patterns are grouped into two categories: strategies or concurrent execution mechanisms. When strategizing to write a program, two things you need to consider are what algorithmic strategy to use and what implementation to use on the algorithmic strategy you picked. There are two major categories for parallel code patterns, process/thread control patterns and coordination patterns. Most software uses message passing or mutual exclusion during parallel processing. Message Passing Interface (MPI) and OpenMP are two types of computation. There's an upcoming parallel implementation involves hybrid computation that uses both patterns mentioned above.

### *5. In the section "Categorizing Patterns" in the "Introduction to Parallel Computing\_3.pdf" compare the following: Collective synchronization (barrier) with Collective communication (reduction) & Master-worker with fork-join*

Collective synchronization: Blocks the process until all the processes have reached a synchronization point.

Collective communication: Reduce the process to smaller processes. All processes must reach a common point before they execute again.

Master-work: Divide the work so multiple machines can work on the process.

Fork-join: Divides the work and make them join again at a certain point

6. *Where can we find parallelism in programming?*

We can find parallelism in programs, data, and resources. In programs you find parallelism between program statements or blocks of codes (block level, loop level, routine level, or processing level). In data, you find it where the data is located or how the data is operated on.

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

Dependency happens when one operation depends on another operation to complete first for it to execute. The types of dependency are true (flow) dependence, output dependence, and anti-dependence.

True dependency:

a=5

b=a

Output dependency:

a=f(x)

a=b

Anti-dependency:

a=b

b=7

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

A statement is independent when the order of execution does not matter.

Ex:

S1: a=3+5;

S2: b=2;

A statement is dependent when the order of execution does matter and affects the output.

Ex:

S1: a=b+5;

S2: b=2;

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

Two statements can be executed in parallel if it is independent of each other.

10. *How can dependency be removed?*

Modify the program by rearranging or eliminating statements to remove the dependency.

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

```
for (i=0; i<100; i++) //loop 1      true dependency
S1: a[i]=i;
For (i=0; i<100; i++) { //loop 2      anti-dependency
S1: a[i]=i;
S2: b[i]=2*i;
}
```

There is no dependency between the statements for both loops because they do not read the same memory location.

You compute dependency by comparing the IN and OUT sets of each node and by showing the iteration space for each statement. Loop 1's iteration space is:  $S1^0 \rightarrow S1^1 \rightarrow S1^2 \rightarrow \dots \rightarrow S1^{100}$

It has true dependency because the  $i$  is read in  $a[i]$  first and then when we are initializing  $a[i]$ .

Loop 2's iteration space is:

$S1^0 \quad S1^1 \quad S1^2 \quad \dots \quad S1^{100}$

$\downarrow \quad \downarrow \quad \downarrow \quad \dots \quad \downarrow$

$S2^0 \quad S2^1 \quad S2^2 \quad \dots \quad S2^{100}$

Loop 2's has true dependency because the  $i$  is read in  $a[i]$  first and then when we are initializing  $a[i]$ . It also have dependency type is anti-dependency because  $b[i]=2*I$  is reading  $i$  after statement 1.

**Parallel programming basics**

Sean Staley

In this assignment, I used parallel programming and the ARM assembly language to compute

$$\int_0^{\pi} \sin(x) dx$$

I did this using the trapezoidal rule with  $2^{20}$  equal subdivisions. The code is shown below. (Has errors)

```

1 //The answer from this computation should
  be 2.0.
2 #include <math.h>
3 #include <stdio.h> // printf()
4 #include <stdlib.h> // atoi()
5 #include <omp.h> // OpenMP
6
7
8 /* Demo program for OpenMP: computes
  trapezoidal approximation to an integral*/
9
10 const double pi =
11 3.141592653589793238462643383079;
12 int main(int argc, char** argv) {
13 /* Variables */
14 double a = 0.0, b = pi; /* limits of
  integration */;
15 int n = 1048576; /* number of
  subdivisions = 2^20 */
16 double h = (b - a) / n; /* width of
  subdivision */
17 double integral; /* accumulates answer */
18 int threadcnt = 1;
19
20 double f(double x);
21
22 /* parse command-line arg for number of
  threads */
23 if (argc > 1) {
24 threadcnt = atoi(argv[1]);
25 }
26
27 #ifdef _OPENMP
28 omp_set_num_threads( threadcnt );
29 printf("OMP defined, threadcnt = %d\n",
  threadcnt);
30 #else
31 printf("OMP not defined");
32 #endif
33
34 integral = (f(a) + f(b))/2.0;
35 int i;
36
37 #pragma omp parallel for private(i)
  shared (a, n, h, integral)
38 for(i = 1; i < n; i++) {
39 integral += f(a+i*h);
40 }
41
42 integral = integral * h;
43 printf("With %d trapezoids, our estimate
  of the integral from \n", n);
44 printf("%f to %f is %f\n", a,b,integral);
45 }
46
47 double f(double x) {
48 return sin(x);
49 }

```

The code above has a problem on line 37. A corrected version is shown below.

```

1 //The answer from this computation should
  be 2.0.
2 #include <math.h>
3 #include <stdio.h> // printf()
4 #include <stdlib.h> // atoi()
5 #include <omp.h> // OpenMP
6
7
8 /* Demo program for OpenMP: computes
  trapezoidal approximation to an integral*/
9
10 const double pi =
11 3.141592653589793238462643383079;
12
13 int main(int argc, char** argv) {
14     /* Variables */
15     double a = 0.0, b = pi; /* limits of
16     integration */;
17     int n = 1048576; /* number of
18     subdivisions = 2^20 */
19     double h = (b - a) / n; /* width of
20     subdivision */
21     double integral; /* accumulates answer */
22     int threadcnt = 1;
23
24     double f(double x);
25
26     /* parse command-line arg for number of
27     threads */
28     if (argc > 1) {
29         threadcnt = atoi(argv[1]);
30     }
31
32     #ifdef _OPENMP
33     omp_set_num_threads( threadcnt );
34     printf("OMP defined, threadcnt = %d\n",
35     threadcnt);
36     #else
37     printf("OMP not defined");
38     #endif
39
40     integral = (f(a) + f(b))/2.0;
41
42     int i;
43     #pragma omp parallel for \
44     private(i) shared (a, n, h) reduction(+:
45     integral)
46     for(i = 1; i < n; i++) {
47         integral += f(a+i*h);
48     }
49
50     integral = integral * h;
51     printf("With %d trapezoids, our estimate
52     of the integral from \n", n);
53     printf("%f to %f is %f\n", a,b,integral);
54 }
55
56 double f(double x) {
57     return sin(x);
58 }
59

```

I used a nano editor to copy and paste the code into the Raspberry Pi to create the necessary files to run this program.

```

pi@raspberrypi: ~
File Edit Tabs Help
GNU nano 2.7.4 File: trap-notworking.c Modified

//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);
}

Get Help Write Out Where Is Cut Text Justify Cur Pos Prev Page First Line
Exit Read File Replace Uncut Text To Spell Go To Line Next Page Last Line

pi@raspberrypi: ~
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> // 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) 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);
}

Read 60 lines
Get Help Write Out Where Is Cut Text Justify Cur Pos Prev Page First Line
Exit Read File Replace Uncut Text To Spell Go To Line Next Page Last Line

```



When I tried to compile these two codes into executables though I ran into a problem.

```
pi@raspberrypi:~ $ gcc trap-notworking.c -o trap-notworking -fopenmp
/tmp/ccybUX3K.o: In function 'f':
trap-notworking.c:(.text+0x17c): undefined reference to `sin'
collect2: error: ld returned 1 exit status
```



The command typed into terminal, “**gcc trap-notworking.c -o trap-notworking -fopenmp**” Should have given me an executable file, but it was returning an error. After some troubleshooting and googling I was able to find that the “sin(x)” function was the problem. I changed “sin(x)” to 0 in both files and ran the code, and it gave me the executable. But this was obviously not the correct answer.

```
pi@raspberrypi:~ $ ./trap-notworking 4
OMP defined, threadct = 4
With 1048576 trapezoids, our estimate of the integral from
0.000000 to 3.141593 is 0.000000
pi@raspberrypi:~ $ ./trap-working 4
OMP defined, threadct = 4
With 1048576 trapezoids, our estimate of the integral from
0.000000 to 3.141593 is 0.000000
pi@raspberrypi:~ $
```

At this point the code was running, but I needed to be able to compute sin(x) in order to arrive at the correct answer. I found online that since the sin function was part of the math family imported in the second line of code, I needed to include a special instruction “-lm” when compiling into an executable in the terminal in order for it to work. I changed “0” back to sign and tried it.

```
pi@raspberrypi:~ $ gcc trap-notworking.c -o trap-notworking -fopenmp -lm
pi@raspberrypi:~ $
pi@raspberrypi:~ $ gcc trap-working.c -o trap-working -fopenmp -lm
pi@raspberrypi:~ $
```

This gave me the correct executable files,

 trap-notworking	8.7 KiB	03/26/2019 19:04
 trap-notworking.c	1.1 KiB	03/26/2019 19:02
 trap-working	14.0 KiB	03/26/2019 19:05
 trap-working.c	1.1 KiB	03/26/2019 19:05

which gave me the correct answers,

```
pi@raspberrypi:~ $ ./trap-notworking 4
OMP defined, threadct = 4
With 1048576 trapezoids, our estimate of the integral from
0.000000 to 3.141593 is 1.370676
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:~ $ □
```

Notice how when we run trap-not working with 4 threads, we get the wrong answer. This is because of the problem in the code on line 37, which was fixed in trap-working. Each thread took on a separate section of the 1048576/4 trapezoids and calculated them simultaneously to arrive at the correct estimate of 2.0.

## ARM Assembler in Raspberry PI

Kishan Bhakta

### Part 1 Source Code

```
pi@raspberrypi: ~  
pi@raspberrypi:~ $ cat fourth.s  
@Fourth program  
@This program compute the following if statement construct:  
    @int x;  
    @int y;  
    @if(x == 0)  
    @ y=1;  
.section .data  
.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  
.global _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  
pi@raspberrypi:~ $
```

Program execution flow:

1. Load the memory address of x into r1.
2. Load the value of x into r1.

3. Compare value in r1 with 0.
4. Branch if true to thenpart.
5. thenpart:
  - a. move 1 into r2.
  - b. Load the memory address of y into r3.
  - c. Load the value of y into r2.
6. endofif:
  - a. exit syscall
  - b. wake kernel

## Part 1 Debug

```

pi@raspberrypi: ~
(gdb) b 14
Breakpoint 1 at 0x10078: file fourth.s, line 14.
(gdb) b 19
Breakpoint 2 at 0x10084: file fourth.s, line 19.
(gdb) run
Starting program: /home/pi/fourth

Breakpoint 1, _start () at fourth.s:15
15      ldr r1,[r1]      @load the value x into r1
(gdb) info registers r1 r2 r3 r7
r1      0x200a4  131236
r2      0x0      0
r3      0x0      0
r7      0x0      0
(gdb) stepi
17      cmp r1,#0      @
(gdb) info registers r1 r2 r3 r7
r1      0x0      0
r2      0x0      0
r3      0x0      0
r7      0x0      0
(gdb) stepi
18      beq thenpart   @branch (jump) if true (Z==1) to the thenpart
(gdb) info registers r1 r2 r3 r7
r1      0x0      0
r2      0x0      0
r3      0x0      0
r7      0x0      0
(gdb) stepi
thenpart () at fourth.s:21
21      mov r2,#1
(gdb) info registers r1 r2 r3 r7
r1      0x0      0
r2      0x0      0
r3      0x0      0
r7      0x0      0
(gdb) stepi
22      ldr r3,y        @load the memory address of y into r3
(gdb) info registers r1 r2 r3 r7
r1      0x0      0
r2      0x1      1
r3      0x0      0
r7      0x0      0
(gdb) stepi
23      ldr r2,[r3]     @load r2 register value into y memory address
(gdb) info registers r1 r2 r3 r7
r1      0x0      0
r2      0x1      1
r3      0x200a8  131240
r7      0x0      0
(gdb) stepi
endofif () at fourth.s:25
25      mov r7,#1      @Program Termination: exit syscall
(gdb) info registers r1 r2 r3 r7
r1      0x0      0

```

```

pi@raspberrypi: ~
17      cmp r1,#0      @
(gdb) info registers r1 r2 r3 r7
r1          0x0      0
r2          0x0      0
r3          0x0      0
r7          0x0      0
(gdb) stepi
18      beq thenpart   @branch (jump) if true (Z==1) to the thenpart
(gdb) info registers r1 r2 r3 r7
r1          0x0      0
r2          0x0      0
r3          0x0      0
r7          0x0      0
(gdb) stepi
thenpart () at fourth.s:21
21      mov r2,#1
(gdb) info registers r1 r2 r3 r7
r1          0x0      0
r2          0x0      0
r3          0x0      0
r7          0x0      0
(gdb) stepi
22      ldr r3,=y      @load the memory address of y into r3
(gdb) info registers r1 r2 r3 r7
r1          0x0      0
r2          0x1      1
r3          0x0      0
r7          0x0      0
(gdb) stepi
23      ldr r2,[r3]    @load r2 register value into y memory address
(gdb) info registers r1 r2 r3 r7
r1          0x0      0
r2          0x1      1
r3          0x200a8   131240
r7          0x0      0
(gdb) stepi
endif () at fourth.s:25
25      mov r7,#1      @Program Termination: exit syscall
(gdb) info registers r1 r2 r3 r7
r1          0x0      0
r2          0x0      0
r3          0x200a8   131240
r7          0x0      0
(gdb) stepi
26      svc #0         @Program Termination: wake kernel
(gdb) info registers r1 r2 r3 r7
r1          0x0      0
r2          0x0      0
r3          0x200a8   131240
r7          0x1      1
(gdb) stepi
[Inferior 1 (process 1633) exited normally]
(gdb) x/1xw 131240
0x200a8:      0x00000000
(gdb)

```

Y memory location: 131240

Zflag:

## Part 2 Source Code

```
pi@raspberrypi: ~  
pi@raspberrypi:~ $ cat fourth.s  
@Fourth program  
@This program compute the following if statement contruct:  
    @intx;  
    @inty;  
    @if(x == 0)  
    @ y=1;  
.section .data  
.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  
.global _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 not true (Z==1) to the thenpart (De Morgans)  
    @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  
pi@raspberrypi:~ $
```

Program execution flow:

1. Load the memory address of x into r1.
2. Load the value of x into r1.
3. Compare value in r1 with 0.
4. Branch if not equal and move straight to endofif.
5. Else

6. thenpatr:
  - a. move 1 into r2.
  - b. Load the memory address of y into r3.
  - c. Load the value of y into r2.
7. endofif:
  - a. exit syscall
  - b. wake kernel

## Part 2 Debug

```

pi@raspberrypi: ~
(gdb) b 14
Breakpoint 1 at 0x10078: file fourth.s, line 14.
(gdb) b 19
Breakpoint 2 at 0x10088: file fourth.s, line 19.
(gdb) run
Starting program: /home/pi/fourth

Breakpoint 1, _start () at fourth.s:15
15      ldr r1,[r1]      @load the value x into r1
(gdb) info registers r1 r2 r3 r7
r1      0x200a0  131232
r2      0x0     0
r3      0x0     0
r7      0x0     0
(gdb) stepi
17      cmp r1,#0      @
(gdb) info registers r1 r2 r3 r7
r1      0x0     0
r2      0x0     0
r3      0x0     0
r7      0x0     0
(gdb) stepi
18      bne endofif    @branch (jump) if not true (Z==1) to the thenpart (De Morgans)
(gdb) info registers r1 r2 r3 r7
r1      0x0     0
r2      0x0     0
r3      0x0     0
r7      0x0     0
(gdb) stepi
thenpart () at fourth.s:21
21      mov r2,#1
(gdb) info registers r1 r2 r3 r7
r1      0x0     0
r2      0x0     0
r3      0x0     0
r7      0x0     0
(gdb) stepi
Breakpoint 2, thenpart () at fourth.s:22
22      ldr r3,y        @load the memory address of y into r3
(gdb) info registers r1 r2 r3 r7
r1      0x0     0
r2      0x1     1
r3      0x0     0
r7      0x0     0
(gdb) stepi
23      ldr r2,[r3]      @load r2 register value into y memory address
(gdb) info registers r1 r2 r3 r7
r1      0x0     0
r2      0x1     1
r3      0x200a4  131236
r7      0x0     0
(gdb) stepi
endofif () at fourth.s:25
25      mov r7,#1        @Program Termination: exit syscall
  
```

```

pi@raspberrypi: ~
r1      0x0      0
r2      0x0      0
r3      0x0      0
r7      0x0      0
(gdb) stepi
18          bne endofif      @branch (jump) if not true (Z==1) to the thenpart (De Morgans)
(gdb) info registers r1 r2 r3 r7
r1      0x0      0
r2      0x0      0
r3      0x0      0
r7      0x0      0
(gdb) stepi
thenpart () at fourth.s:21
21          mov r2,#1
(gdb) info registers r1 r2 r3 r7
r1      0x0      0
r2      0x0      0
r3      0x0      0
r7      0x0      0
(gdb) stepi

Breakpoint 2, thenpart () at fourth.s:22
22          ldr r3,=y      @load the memory address of y into r3
(gdb) info registers r1 r2 r3 r7
r1      0x0      0
r2      0x1      1
r3      0x0      0
r7      0x0      0
(gdb) stepi
23          ldr r2,[r3]      @load r2 register value into y memory address
(gdb) info registers r1 r2 r3 r7
r1      0x0      0
r2      0x1      1
r3      0x200a4    131236
r7      0x0      0
(gdb) stepi
endofif () at fourth.s:25
25          mov r7,#1      @Program Termination: exit syscall
(gdb) info registers r1 r2 r3 r7
r1      0x0      0
r2      0x0      0
r3      0x200a4    131236
r7      0x0      0
(gdb) stepi
26          svc #0      @Program Termination: wake kernel
(gdb) info registers r1 r2 r3 r7
r1      0x0      0
r2      0x0      0
r3      0x200a4    131236
r7      0x1      1
(gdb) stepi
[Inferior 1 (process 1779) exited normally]
(gdb) x/1xw 131236
0x200a4: 0x00000000
(gdb)

```

Y memory location: 131236

Zflag:

### Part 3 Source Code



```

pi@rasberrypi: ~
pi@rasberrypi:~ $ cat ControlStructure.s
@ControlStructure1
@This program compute the following if statement contract:
    @if x<=3;
    @x=x-1;
    @else;
    @x=x-2
    @x = 1;
.section .data
.data
x: .word 4 @ 32-bit signed integer, you can also use int directive instead of .word directive

.section .text
.global _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 less than or equal to 3
    sub r1,r1,#2    @subtract 2 from r1 and store in r1
    b endofif      @jump to endofif

thenpart:
    sub r1,r1,#1    @subtract 1 from r1 and store into r1
endofif:

    mov r7,#1      @Program Termination: exit syscall
    svc #0         @Program Termination: wake kernel
.end
pi@rasberrypi:~ $ █

```

Program execution flow:

1. Load the memory address of x into r1.
2. Load the value of x into r1.
3. Compare value in r1 with 3.
4. Branch if less than or equal to 3.
  - a. Answer does not match pass condition because  $x = 4$ .
5. Subtract 2 from r1 and store into r1
6. Jump to endofif

7. endofif:
  - a. exit syscall
  - b. wake kernel

### Part 3 Debug

```

pi@raspberrypi: ~
22
23     thenpart:
24         sub r1,r1,#1    @subtract 1 from r1 and store into r1
25     endofif:
26
27         mov r7,#1       @Program Termination: exit syscall
28         svc #0          @Program Termination: wake kernel
29     .end
(gdb)
Line number 30 out of range; ControlStructure.s has 29 lines.
(gdb) b 17
Breakpoint 1 at 0x1007c: file ControlStructure.s, line 17.
(gdb) run
Starting program: /home/pi/ControlStructure

Breakpoint 1, _start () at ControlStructure.s:18
18         cmp r1,#3      @
(gdb) info registers r1 r2 r7
r1          0x4          4
r2          0x0          0
r7          0x0          0
(gdb) stepi
19         ble thenpart   @branch (jump) if less than or equal to 3
(gdb) info registers r1 r2 r7
r1          0x4          4
r2          0x0          0
r7          0x0          0
(gdb) stepi
20         sub r1,r1,#2    @subtract 2 from r1 and store in r1
(gdb) info registers r1 r2 r7
r1          0x4          4
r2          0x0          0
r7          0x0          0
(gdb) stepi
21         b endofif      @jump to endofif
(gdb) info registers r1 r2 r7
r1          0x2          2
r2          0x0          0
r7          0x0          0
(gdb) stepi
endofif () at ControlStructure.s:27
27         mov r7,#1       @Program Termination: exit syscall
(gdb) info registers r1 r2 r7
r1          0x2          2
r2          0x0          0
r7          0x0          0
(gdb) stepi
28         svc #0          @Program Termination: wake kernel
(gdb) info registers r1 r2 r7
r1          0x2          2
r2          0x0          0
r7          0x1          1
(gdb) x/lxw 0x1007c
0x1007c <_start+8>:    0xe3510003
(gdb)

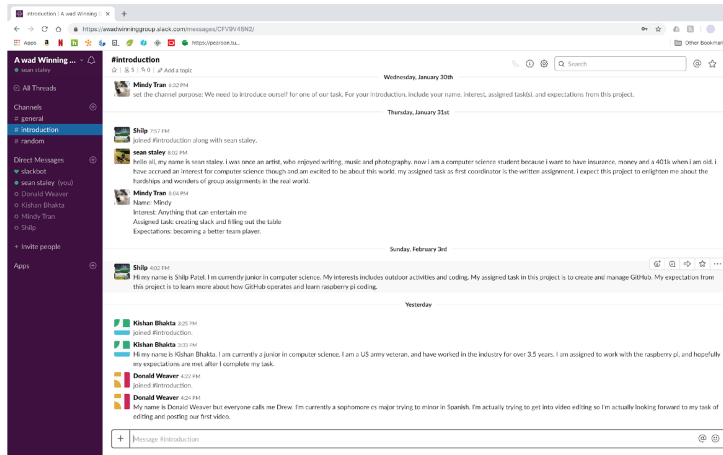
```

Y memory location: 0x1007c

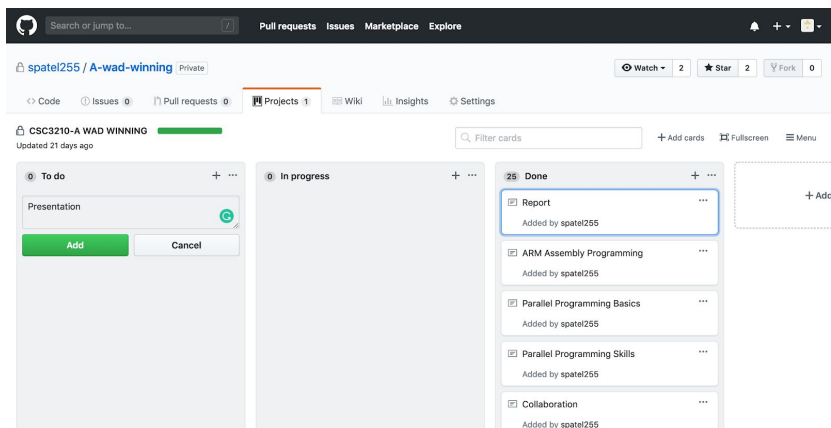
Zflag:

## Appendix

Slack - <https://awadwinninggroup.slack.com/>



Github - <https://github.com/spatel255/A-wad-winning/>



YouTube Channel - [https://www.youtube.com/channel/UCVEvyRC9\\_1g\\_8306r68504g](https://www.youtube.com/channel/UCVEvyRC9_1g_8306r68504g)