

DEVELOPING SOFT AND PARALLEL PROGRAMMING SKILLS USING PROJECT BASED LEARNING

CSC-3210 SPRING 2019 (M/W-CLASS)

INSTRUCTOR: DR. AWAD MUSSA

DATE: 03/08/2019

SUBMITTED BY TEAM HACKS
DAVID CARR
JEFFERY KONG
MOHAMED ALI
SABINA MAHARJAN
VINCENT HU

PLANNING AND SCHEDULING:

Assignee Name	Email	Task	Duration (hours)	Dependency	Due date
David Carr	Dcarr15@student.gsu.edu	Task 1	1hr		3/2/19
Jeffery Kong	Jkong8@student.gsu.edu	Programmer	3h	Must have PI	3/4/19
Mohamed Ali	Mali41@student.gsu.edu	Group coordinator	2h	Read and answer questions	3/3/19
Sabina Maharjan	Smaharjan1@student.gsu.edu	Report	2h	Slack, GitHub	3/3/19
Vincent Hu	Vhu2@student.gsu.edu	Programmer	3h	Must have PI	3/4/19
Everyone		Record Video			

PARALLEL PROGRAMMING SKILLS:

Part A: Foundation

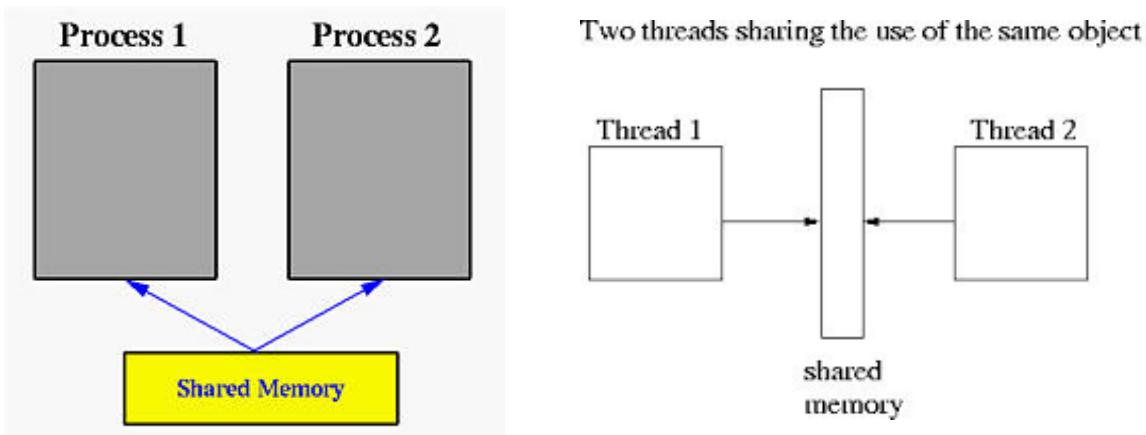
- Define the following: Task, Pipelining, Shared Memory, Communications, Synchronization. (in your own words)
 - A task is a set of instructions that are executed by the processor and parallel program consists of many tasks.
 - Pipelining is allowing multiple instructions to be processed at the same time.
 - Shared Memory is a form of memory where all processors have direct access to a shared physical memory and parallel tasks can directly access that same logical memory.
 - Communications is a form of data exchange between parallel tasks.
 - Synchronization is a point where a task may not proceed further until another task reaches that same point.
- Classify parallel computers based on Flynn's taxonomy. Briefly describe every one of them
 - Single Instruction, Single Data (SISD): executes exactly one instruction stream at a time

- Single Instruction, Multiple Data (SIMD): is an instruction set architecture (ISA) that have a single CU and more than one PU, and it executes a single instruction stream over PU handled through the CU.
- Multiple Instruction, Single Data (MISD): is an ISA for parallel computing where multiple functional units execute different operations on the same data set.
- Multiple Instruction, Multiple Data (MIMD): is an ISA for parallel computing that is ideal for computers with multiprocessors and each processor may be executing a different instruction stream.

- What are the Parallel Programming Models?
- Shared Memory, Threads, Distributed Memory, Data Parallel, Hybrid, Single Program Multiple Data (SPMD), Multiple Program Multiple Data (MPMD)

- List and briefly describe the types of Parallel Computer Memory Architectures. What type is used by OpenMP and why?
- General Characteristics: multiple processors that can run separately but share the same memory and all the processors have access to the memory as global address space.
- Uniform Memory Access (UMA): identical processors that have equal access times to memory and upon update on location in shared memory, all other processors know about the update.
- Non-Uniform Memory Access (NUMA): SMPs are linked, and SMP can directly access the memory of another SMP.
- Uniform Memory Access uses OpenMP because OpenMP targets shared-memory architectures.

- Compare Shared Memory Model with Threads Model? (in your own words and show pictures)



- Shared Memory is the simplest parallel programming model, and tasks share a common address space, without threads Model it's hard to manage data locality.

- What is Parallel Programming? (in your own words)
- Parallel Programming is using multiple computing resources to solve problem concurrently.

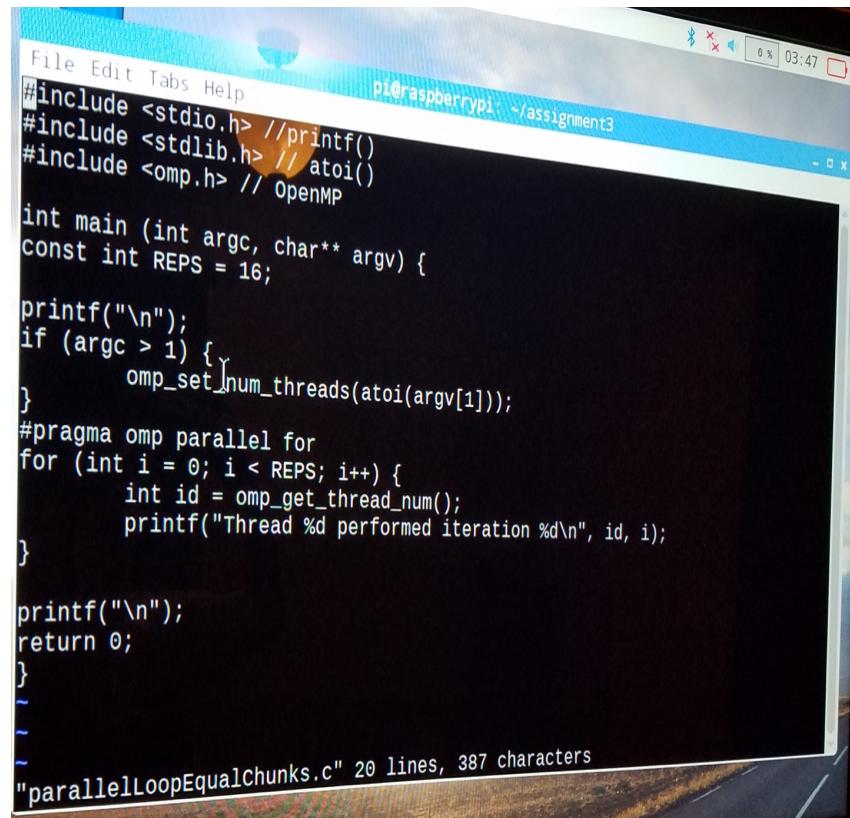
- What is system on chip (SoC)? Does Raspberry PI use system on SoC?
- SoC integrates all the components into a single chip.
- Yes, Raspberry PI uses system on SoC.

- Explain what the advantages are of having a System on a Chip rather than separate CPU, GPU and RAM components
- System on a chip consumes less power and costs less. Also, System on a chip is small and lightweight compared to having separate CPU and RAM.

Part B: Parallel Programming Basics

After reading through the Parallel Programming Task assignment, we started to copy the example code onto our Raspberry Pi.

Our understanding of the code is that the line: `#pragma omp parallel for`, is different from the previous project's code: `#pragma omp parallel` because it specifically tells the processor to run the for loop in parallel, not just all of the code in the block, so the for loop is split up among the 4 processors of the Raspberry Pi.



```

File Edit Tabs Help pi@raspberrypi: ~/assignment3
#include <stdio.h> //printf()
#include <stdlib.h> // atoi()
#include <omp.h> // OpenMP

int main (int argc, char** argv) {
const int REPS = 16;

printf("\n");
if (argc > 1) {
    omp_set_num_threads(atoi(argv[1]));
}

#pragma omp parallel for
for (int i = 0; i < REPS; i++) {
    int id = omp_get_thread_num();
    printf("Thread %d performed iteration %d\n", id, i);
}

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

"parallelLoopEqualChunks.c" 20 lines, 387 characters

```

After copying the example code, we had to compile and link it using the gcc command. This command took the source code: parallelLoopsEqualChunks.c and created an executable called pLoop. In order to run the program, We had to call ./pLoop with the optional argument of 4.

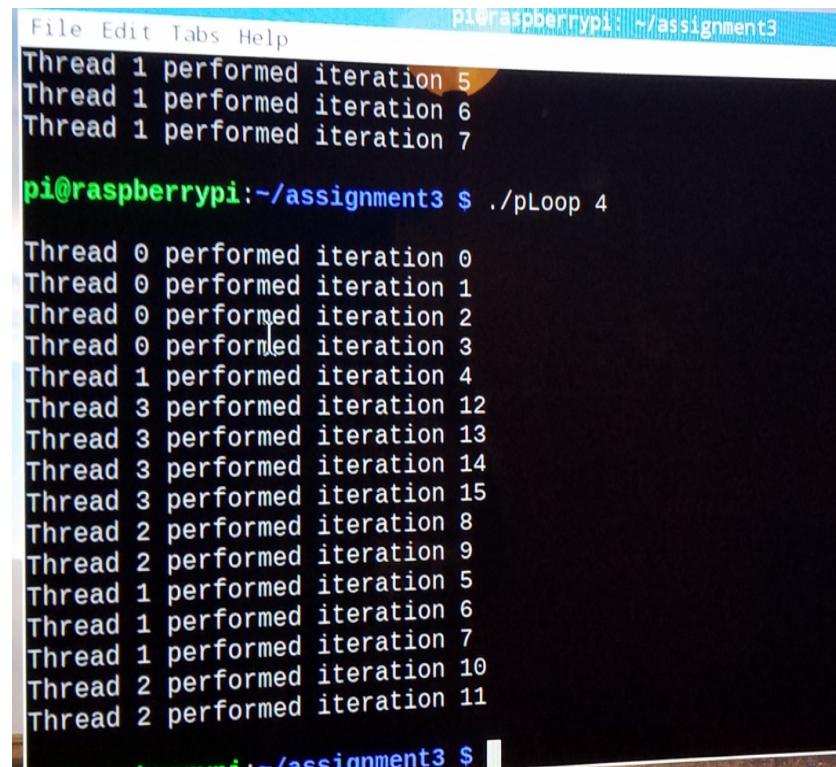
```
pi@raspberrypi:~/assignment3
pi@raspberrypi:~/assignment3 $ vi parallelLoopEqualChunks.c
pi@raspberrypi:~/assignment3 $ gcc parallelLoopEqualChunks.c -o pLoop -fopenmp
pi@raspberrypi:~/assignment3 $
```

At first, we tried to run the command using pLoop 4, but because the ./ path is not in our shell's current \$PATH variable, the command was not found. We could have added the ./ path to the \$PATH variable but it was easier to simply add the directory to the command name. The output split the 16 loops among the 4 processors and ran them in parallel. So, each thread performed 4 iterations.

```
pi@raspberrypi:~/assignment3
pi@raspberrypi:~/assignment3 $ ls
parallelLoopEqualChunks.c  pLoop
pi@raspberrypi:~/assignment3 $ pLoop 4
bash: pLoop: command not found
pi@raspberrypi:~/assignment3 $ ./pLoop 4

Thread 0 performed iteration 0
Thread 0 performed iteration 1
Thread 0 performed iteration 2
Thread 0 performed iteration 3
Thread 3 performed iteration 12
Thread 3 performed iteration 13
Thread 3 performed iteration 14
Thread 3 performed iteration 15
Thread 2 performed iteration 8
Thread 2 performed iteration 9
Thread 2 performed iteration 10
Thread 2 performed iteration 11
Thread 1 performed iteration 4
Thread 1 performed iteration 5
Thread 1 performed iteration 6
Thread 1 performed iteration 7
pi@raspberrypi:~/assignment3 $
```

To double check, command prompt was checked again for the output. It is worth noting that the order that the iterations completed was random.

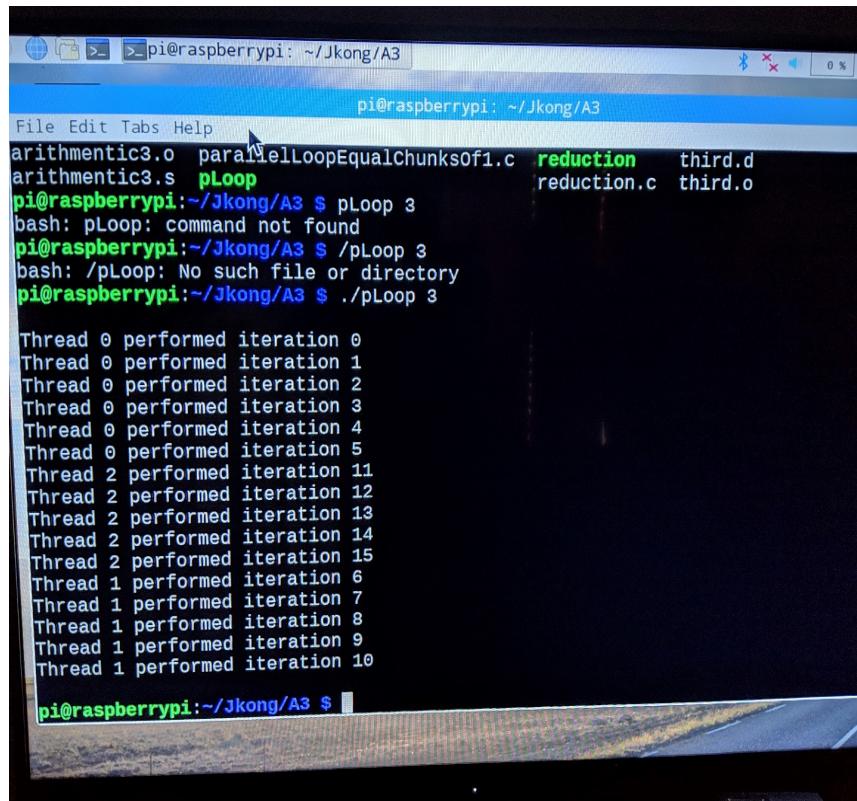


```

pi@raspberrypi:~/assignment3$ ./pLoop 4
Thread 1 performed iteration 5
Thread 1 performed iteration 6
Thread 1 performed iteration 7
Thread 0 performed iteration 0
Thread 0 performed iteration 1
Thread 0 performed iteration 2
Thread 0 performed iteration 3
Thread 1 performed iteration 4
Thread 3 performed iteration 12
Thread 3 performed iteration 13
Thread 3 performed iteration 14
Thread 3 performed iteration 15
Thread 2 performed iteration 8
Thread 2 performed iteration 9
Thread 1 performed iteration 5
Thread 1 performed iteration 6
Thread 1 performed iteration 7
Thread 2 performed iteration 10
Thread 2 performed iteration 11

```

When the program is run with a number that is not evenly divisible by the number of loops, the processor tries to spread the tasks out as best it can. It will divide up the loops by the number of threads and the remainder is added to thread 0.

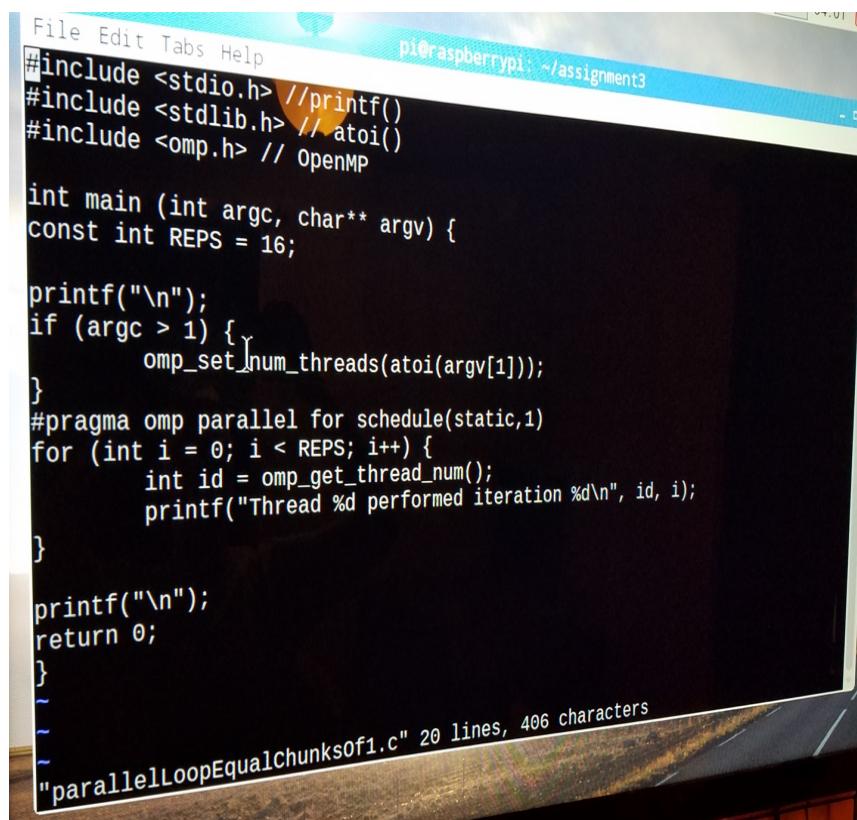


```

pi@raspberrypi:~/Jkong/A3$ pLoop 3
bash: pLoop: command not found
pi@raspberrypi:~/Jkong/A3$ ./pLoop 3
bash: ./pLoop: No such file or directory
pi@raspberrypi:~/Jkong/A3$ ./pLoop 3
Thread 0 performed iteration 0
Thread 0 performed iteration 1
Thread 0 performed iteration 2
Thread 0 performed iteration 3
Thread 0 performed iteration 4
Thread 0 performed iteration 5
Thread 2 performed iteration 11
Thread 2 performed iteration 12
Thread 2 performed iteration 13
Thread 2 performed iteration 14
Thread 2 performed iteration 15
Thread 1 performed iteration 6
Thread 1 performed iteration 7
Thread 1 performed iteration 8
Thread 1 performed iteration 9
Thread 1 performed iteration 10

```

Following the example of parallelism is assigning the threads as each completes its task, so the assignment is done statically. Each thread still has equal amounts of work, just not consecutive iterations.



```

File Edit Tabs Help pi@raspberrypi: ~/assignment3
#include <stdio.h> //printf()
#include <stdlib.h> // atoi()
#include <omp.h> // OpenMP

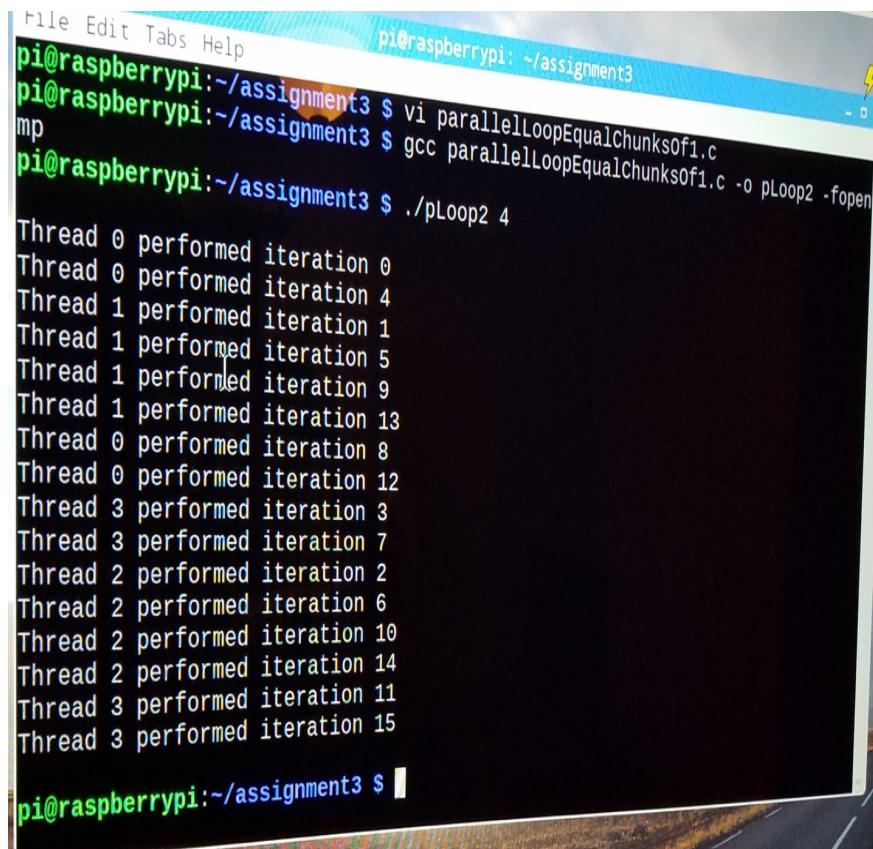
int main (int argc, char** argv) {
const int REPS = 16;

printf("\n");
if (argc > 1) {
    omp_set_num_threads(atoi(argv[1]));
}
#pragma omp parallel for schedule(static,1)
for (int i = 0; i < REPS; i++) {
    int id = omp_get_thread_num();
    printf("Thread %d performed iteration %d\n", id, i);
}

printf("\n");
return 0;
}
-
-
"parallelLoopEqualChunksOf1.c" 20 lines, 406 characters

```

After copying the code, we compiled it and ran it, using the same techniques as an example 1 above. The output was relatively similar between the two examples, even though the second example used a different method of assigning threads.

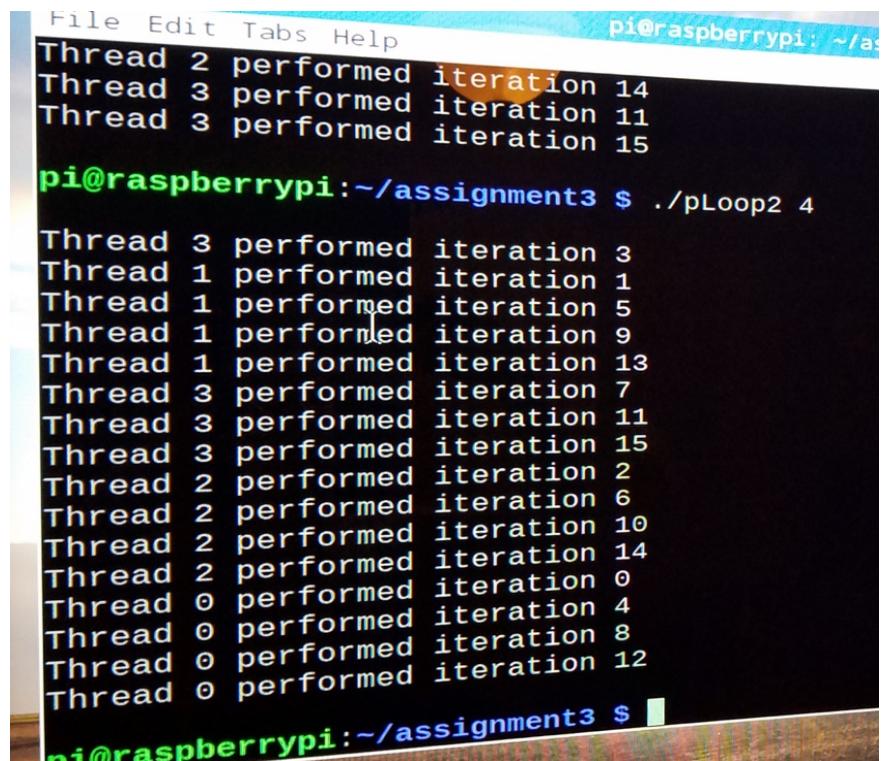


```

File Edit Tabs Help pi@raspberrypi: ~/assignment3
pi@raspberrypi:~/assignments3 $ vi parallelLoopEqualChunksOf1.c
pi@raspberrypi:~/assignments3 $ gcc parallelLoopEqualChunksOf1.c -o pLoop2 -fopen
pi@raspberrypi:~/assignments3 $ ./pLoop2 4
Thread 0 performed iteration 0
Thread 0 performed iteration 4
Thread 1 performed iteration 1
Thread 1 performed iteration 5
Thread 1 performed iteration 9
Thread 1 performed iteration 13
Thread 0 performed iteration 8
Thread 0 performed iteration 12
Thread 3 performed iteration 3
Thread 3 performed iteration 7
Thread 2 performed iteration 2
Thread 2 performed iteration 6
Thread 2 performed iteration 10
Thread 2 performed iteration 14
Thread 3 performed iteration 11
Thread 3 performed iteration 15
pi@raspberrypi:~/assignments3 $

```

We ran the code again, just to ensure that the code was functioning as expected.



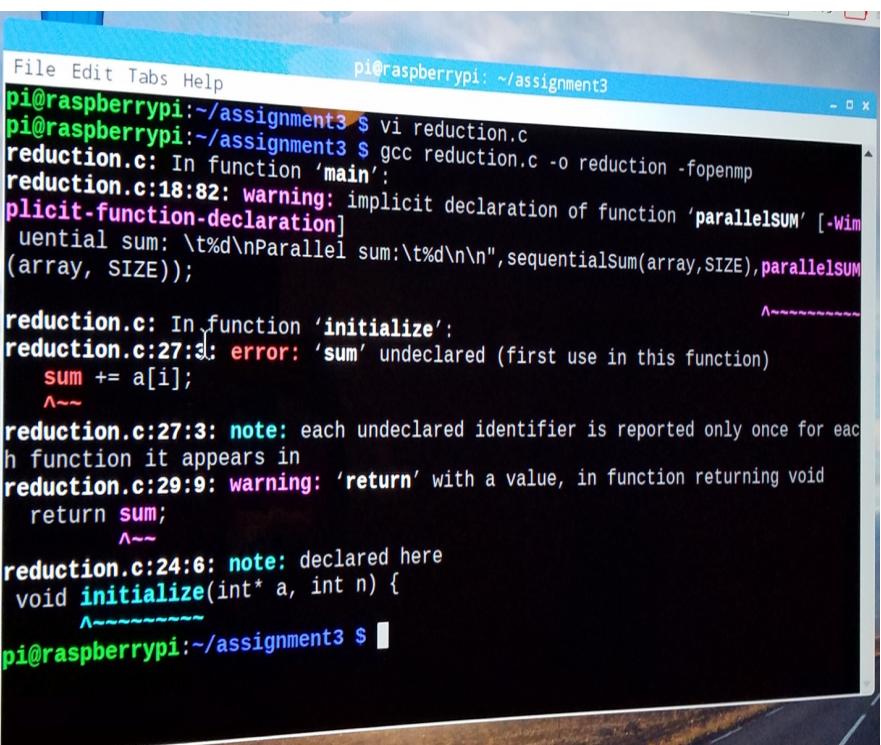
```

File Edit Tabs Help pi@raspberrypi: ~/assignments3
Thread 2 performed iteration 14
Thread 3 performed iteration 14
Thread 3 performed iteration 11
Thread 3 performed iteration 15
pi@raspberrypi:~/assignments3 $ ./pLoop2 4
Thread 3 performed iteration 3
Thread 1 performed iteration 1
Thread 1 performed iteration 5
Thread 1 performed iteration 9
Thread 1 performed iteration 13
Thread 3 performed iteration 7
Thread 3 performed iteration 11
Thread 3 performed iteration 15
Thread 2 performed iteration 2
Thread 2 performed iteration 6
Thread 2 performed iteration 10
Thread 2 performed iteration 14
Thread 2 performed iteration 0
Thread 0 performed iteration 4
Thread 0 performed iteration 8
Thread 0 performed iteration 12
Thread 0 performed iteration 12
pi@raspberrypi:~/assignments3 $

```

The code in example 4.1 Loops with Dependencies was long and difficult to copy, so we had a few errors for the first time we tried to compile the code.

After going back and fixing the errors, we were able to get the code to compile correctly.



```

File Edit Tabs Help pi@raspberrypi: ~/assignment3
pi@raspberrypi:~/assignments3 $ vi reduction.c
pi@raspberrypi:~/assignments3 $ gcc reduction.c -o reduction -fopenmp
reduction.c: In function 'main':
reduction.c:18:82: warning: implicit declaration of function 'parallelSUM' [-Wimplicit-function-declaration]
    sequential sum: \t%d\nParallel sum:\t%d\n\n", sequentialSum(array,SIZE), parallelSUM
(array, SIZE));

reduction.c: In function 'initialize':
reduction.c:27:3: error: 'sum' undeclared (first use in this function)
    sum += a[i];
    ^
reduction.c:27:3: note: each undeclared identifier is reported only once for each
function it appears in
reduction.c:29:9: warning: 'return' with a value, in function returning void
    return sum;
    ^
reduction.c:24:6: note: declared here
void initialize(int* a, int n) {
    ^
pi@raspberrypi:~/assignments3 $

```

In this first example, the `// #pragma omp parallel for // reduction(+:sum)` code is commented out, so the code is not actually running in parallel. When it is not running in parallel, the sums match.

```

File Edit Tabs Help pi@raspberrypi:~/assignments3
pi@raspberrypi:~/assignments3 $ vi reduction.c
pi@raspberrypi:~/assignments3 $ gcc reduction.c -o reduction -fopenmp
pi@raspberrypi:~/assignments3 $ ./reduction 4
Sequential sum: 499562283
Parallel sum: 499562283

pi@raspberrypi:~/assignments3 $ ./reduction 4
Sequential sum: 499562283
Parallel sum: 499562283

pi@raspberrypi:~/assignments3 $ ./reduction 4
Sequential sum: 499562283
Parallel sum: 499562283

pi@raspberrypi:~/assignments3 $ ./reduction 4
Sequential sum: 499562283
Parallel sum: 499562283

```

In this second example, the `#pragma omp parallel for // reduction(+:sum)` code is partially commented. The for loop is still being run in parallel, but the reduction for the sum function is not being called. When a loop with dependencies is being run in parallel but the dependency is not addressed, the results for the function can be unexpected or incorrect. Notice that the Sequential sum and parallel sum do not match.

```

File Edit Tabs Help pi@raspberrypi:~/assignments3
pi@raspberrypi:~/assignments3 $ ./reduction 4
pi@raspberrypi:~/assignments3 $ gcc reduction.c -o reduction -fopenmp
pi@raspberrypi:~/assignments3 $ ./reduction 4
Sequential sum: 499562283
Parallel sum: 152108056

pi@raspberrypi:~/assignments3 $ ./reduction 4
Sequential sum: 499562283
Parallel sum: 153508663

pi@raspberrypi:~/assignments3 $ ./reduction 4
Sequential sum: 499562283
Parallel sum: 153565279

pi@raspberrypi:~/assignments3 $ ./reduction 4
Sequential sum: 499562283
Parallel sum: 152600004

```

In this final example, the code is running in parallel with reduction on the sum function. With the reduction function forcing the threads to run each sum independently and then summing up the final result at the end.

```

pi@raspberrypi:~/assignments3 $ ./reduction 4
Sequential sum: 499562283
Parallel sum: 499562283
pi@raspberrypi:~/assignments3 $ vi reduction.c
pi@raspberrypi:~/assignments3 $ gcc reduction.c -o reduction -fopenmp
pi@raspberrypi:~/assignments3 $ ./reduction 4
Sequential sum: 499562283
Parallel sum: 499562283
pi@raspberrypi:~/assignments3 $ ./reduction 4
Sequential sum: 499562283
Parallel sum: 499562283
pi@raspberrypi:~/assignments3 $ ./reduction 4
Sequential sum: 499562283
Parallel sum: 499562283
pi@raspberrypi:~/assignments3 $ ./reduction 4
Sequential sum: 499562283
Parallel sum: 499562283
pi@raspberrypi:~/assignments3 $

```

With this example, the dependencies could be split up between the threads and totaled at the end of each thread. There are probably some situations where the dependency is not so easily split and recombined, but we will possibly learn about these problems in a future project.

ARM Assembly Programming:

Part 1: Third Program

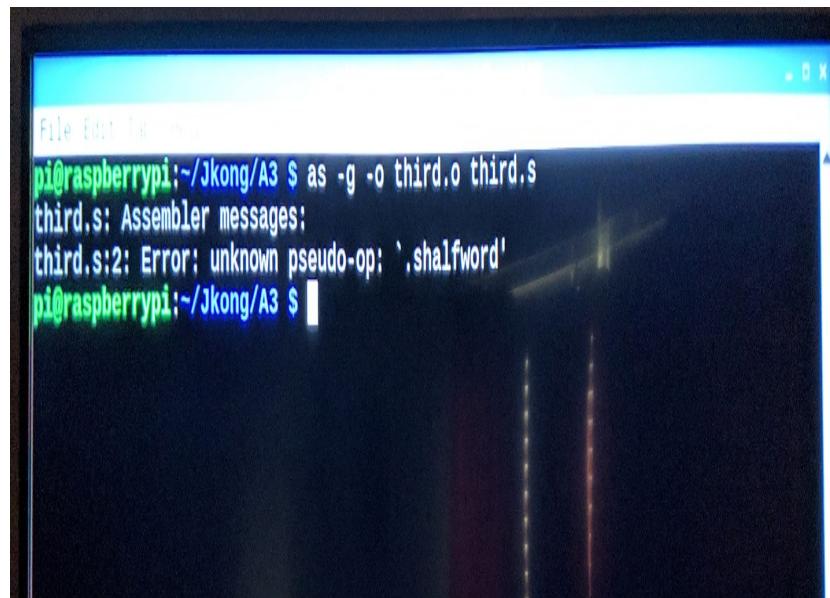
Here is how the specified “third.s” file was created using the vi editor.

```

pi@raspberrypi: ~/Jiang/AS
File Edit Tabs Help
.section.data
a: .shalfword -2 @16bit signed integer
.section .text
.globl _start
_start:
    mov r0, #0x1      @=1
    mov r1, #0xFFFFFFFF @=-1 (signed)
    mov r2, #0xFF      @=255
    mov r3, #0x101     @=257
    mov r4, #0x400     @=1024
    mov r7, #1
.end
~
~
~

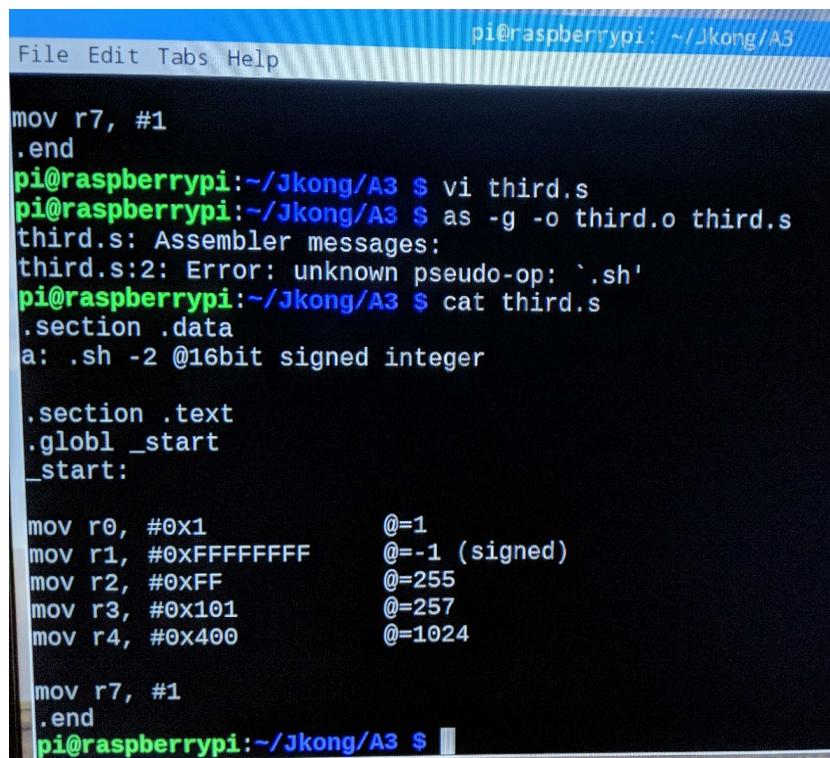
```

After creating the file and assembling it, we ran into an error which states: “unknown pseudo-op: ‘.shalfword’”. We then went back into vi to attempt to fix it.



```
File Edit Tabs Help
pi@raspberrypi:~/Jkong/A3 $ as -g -o third.o third.s
third.s: Assembler messages:
third.s:2: Error: unknown pseudo-op: `shalfword'
pi@raspberrypi:~/Jkong/A3 $
```

We tried to make variable “a” into a 16 bit signed integer was to change the .shalfword to .sh or to .shword but none of those work. Here is the output of showing that the assembler did not recognize .sh for a signed 16 bit integer.



```
File Edit Tabs Help
pi@raspberrypi:~/Jkong/A3 $ vi third.s
pi@raspberrypi:~/Jkong/A3 $ as -g -o third.o third.s
third.s: Assembler messages:
third.s:2: Error: unknown pseudo-op: `sh'
pi@raspberrypi:~/Jkong/A3 $ cat third.s
.section .data
a: .sh -2 @16bit signed integer

.section .text
.globl _start
_start:

    mov r0, #0x1          @=1
    mov r1, #0xFFFFFFFF   @=-1 (signed)
    mov r2, #0xFF          @=255
    mov r3, #0x101         @=257
    mov r4, #0x400         @=1024

    mov r7, #1
.end
pi@raspberrypi:~/Jkong/A3 $
```

After looking through online resources to solve this error, we came to the conclusion that to declare a variable as 16-bit, we have to use “.hword” instead of “.shalfword”. When we load the variable into a register, we can then use “ldrsh” to load the variable as signed.

```
File Edit Tabs Help
.section .data
a: .hword -2 @16bit signed integer

.section .text
.globl _start
_start:

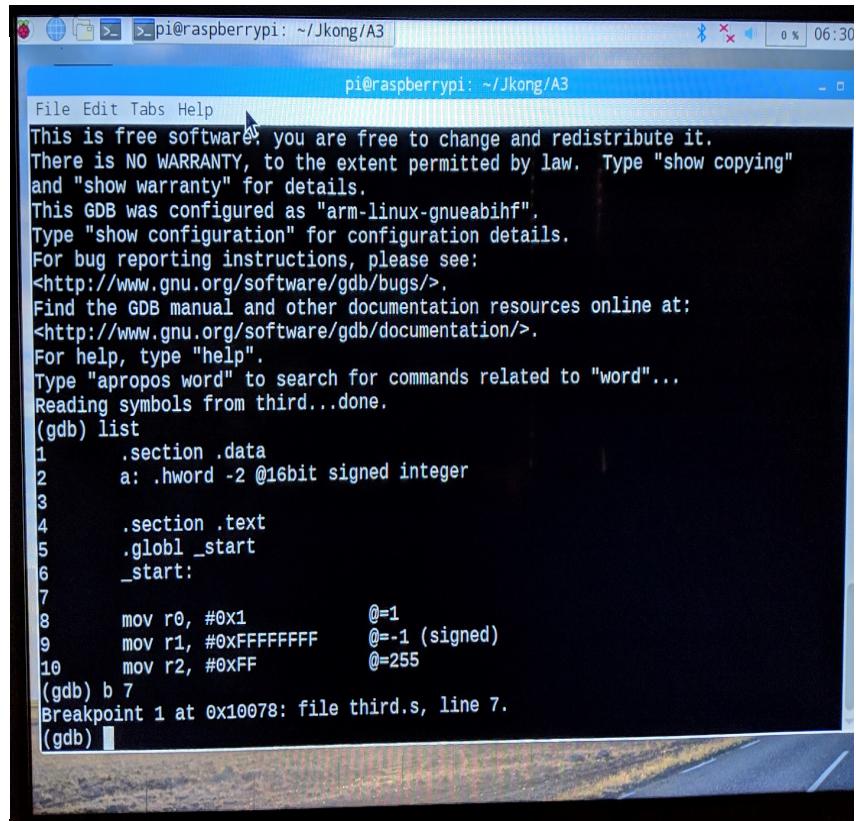
mov r0, #0x1          @=1
mov r1, #0xFFFFFFFF   @=-1 (signed)
mov r2, #0xFF          @=255
mov r3, #0x101         @=257
mov r4, #0x400         @=1024

mov r7, #1
.end
?

"third.s" 15 lines, 224 characters
```

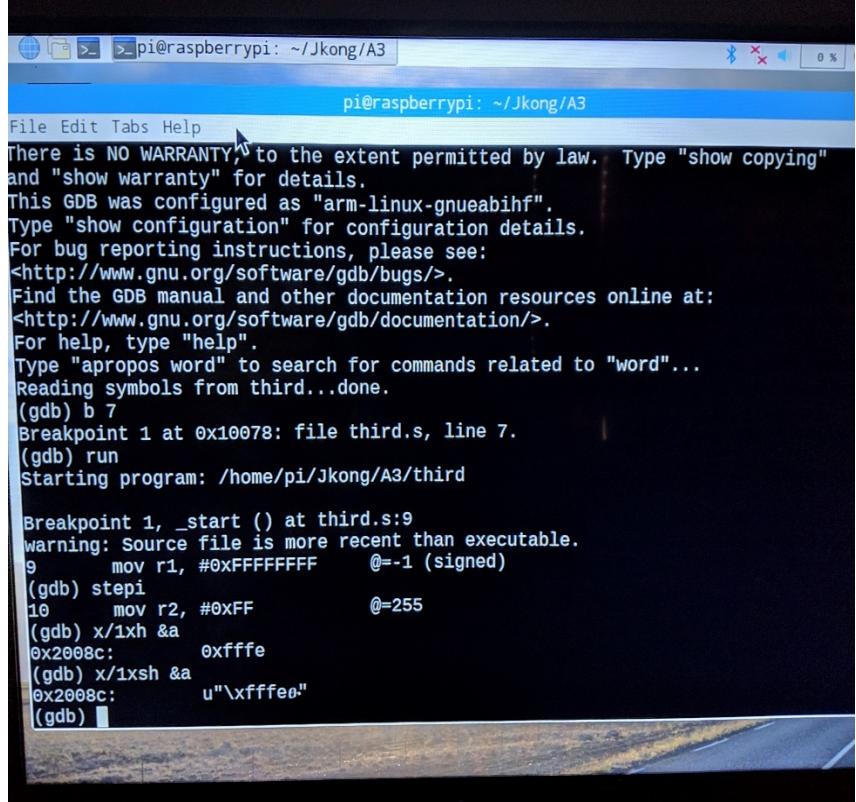
After resolving the error, we were able to assemble the objective file and link it for the executable. We also used the “ls” command to make sure it was there. The “third.d” was a type for when we tried to assemble the file the first time.

```
pi@raspberrypi: ~/Jkong/A3
File Edit Tabs Help
pi@raspberrypi:~/Jkong/A3 $ as -g -o third.d third.s
pi@raspberrypi:~/Jkong/A3 $ ld -o third third.o
pi@raspberrypi:~/Jkong/A3 $ ls
parallelLoopEqualChunks.c    pLoop   reduction   third   third.o
parallelLoopEqualChunksOf1.c pLoop2   reduction.c third.d third.s
pi@raspberrypi:~/Jkong/A3 $
```



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 third...done.
(gdb) list
1 .section .data
2 a: .hword -2 @16bit signed integer
3
4 .section .text
5 .globl _start
6 _start:
7
8 mov r0, #0x1 @=1
9 mov r1, #0xFFFFFFFF @=-1 (signed)
10 mov r2, #0xFF @=255
(gdb) b 7
Breakpoint 1 at 0x10078: file third.s, line 7.
(gdb)

Here is us executing “third” in GDB and setting a break point at line 7.

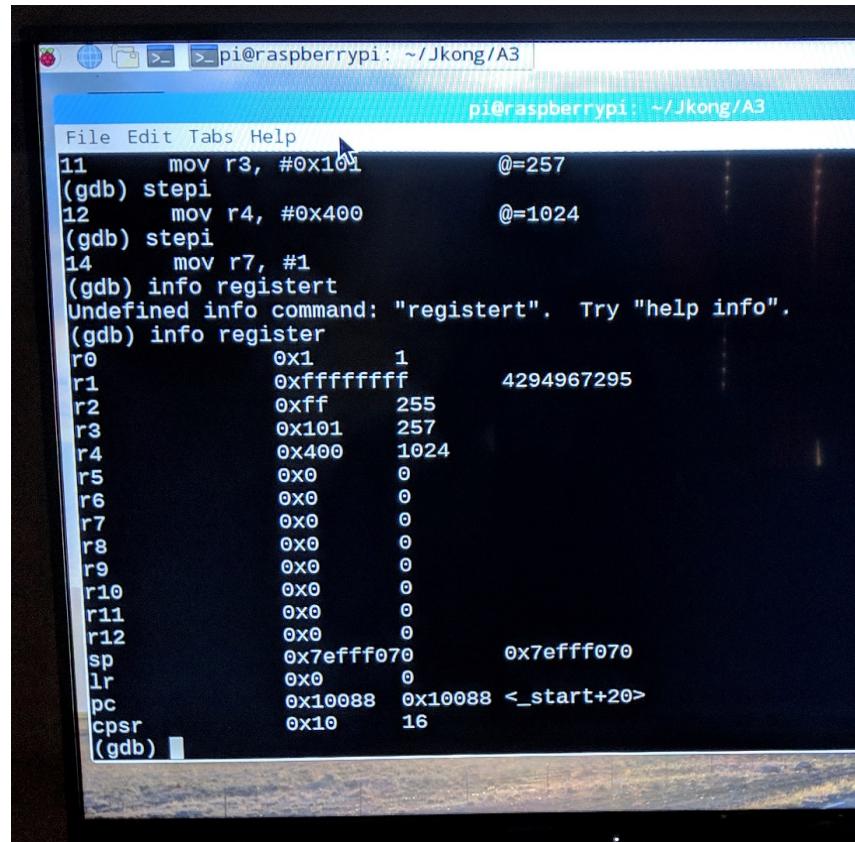


File Edit Tabs Help
 pi@raspberrypi: ~/Jkong/A3
 There is NO WARRANTY, to the extent permitted by law. Type "show copying"
 and "show warranty" for details.
 This GDB was configured as "arm-linux-gnueabihf".
 Type "show configuration" for configuration details.
 For bug reporting instructions, please 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 third...done.
(gdb) b 7
Breakpoint 1 at 0x10078: file third.s, line 7.
(gdb) run
Starting program: /home/pi/Jkong/A3/third

Breakpoint 1, _start () at third.s:9
warning: Source file is more recent than executable.
9 mov r1, #0xFFFFFFFF @=-1 (signed)
(gdb) stepi
10 mov r2, #0xFF @=255
(gdb) x/1xh &a
0x2008c: 0xffffe
(gdb) x/1xsh &a
0x2008c: u"\xffffe"
(gdb)

After setting the breakpoint, we ran the program and used the “x” command to show what the value of the variable “a” is by using “&a” in the command line. When using “x/1xh”, the gdb outputted 0xffffe, which is the 2’s complement representation of -2. However, when using “x/1xsh”, the gdb outputted the same thing along with some other symbols.

Finally, we used the “stepi” command to run through the program and viewed the registers to make sure everything was moved correctly. For register “R1”, the value is 0xffffffff which makes sense because that’s the 2’s complement representation of -1.



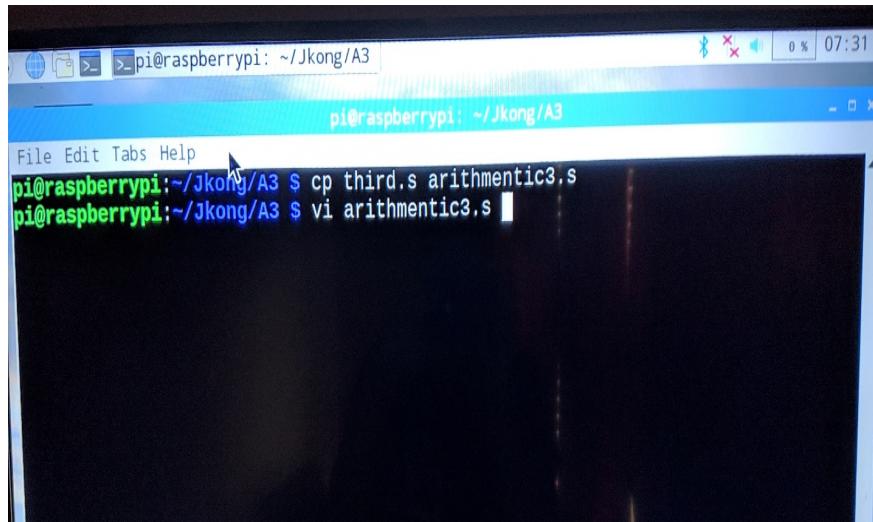
```

pi@raspberrypi: ~/Jkong/A3
File Edit Tabs Help
11    mov r3, #0x101      @=257
(gdb) stepi
12    mov r4, #0x400      @=1024
(gdb) stepi
14    mov r7, #1
(gdb) info register
Undefined info command: "registert". Try "help info".
(gdb) info register
r0          0x1          1
r1          0xffffffff   4294967295
r2          0xff          255
r3          0x101         257
r4          0x400         1024
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          0x7efff070   0x7efff070
lr          0x0           0
pc          0x10088       0x10088 <_start+20>
cpsr        0x10          16
(gdb)

```

Part 2: arithmetic3

To save us the trouble of writing some of the assembly code again, we copied “third.s” from the previous part and used it as a template for writing “arithmetic3.s”. We realized there’s a typo in the picture. Unfortunately, that is the name of my program now.

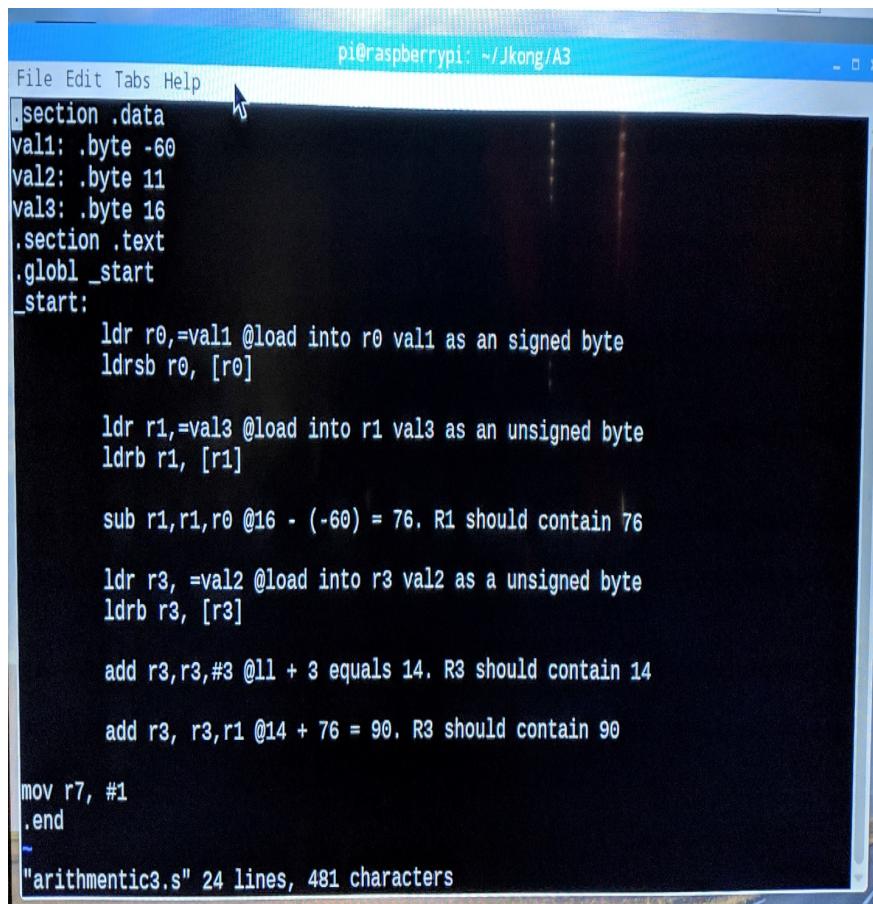


```

pi@raspberrypi:~/Jkong/A3
File Edit Tabs Help
pi@raspberrypi:~/Jkong/A3 $ cp third.s arithmetic3.s
pi@raspberrypi:~/Jkong/A3 $ vi arithmetic3.s

```

Here is the finalized version of my “arithmetic3.s” file. We had trouble trying to load the variables into the registers simply because we did not remember, so we looked back at the arithmetic file from assignment A2. We declared all three variables as a byte and assigned them their variables. Qw solved this expression from right to left. First, we assigned R0 val1 and loaded it as a signed byte. Then, we loaded R1 with val3 as an unsigned byte. We subtracted R1 from R0 and put the result in R1. We then loaded R3 with val2 as an unsigned byte and added 3 to it. All that’s left is to add R3 and R1 and the final answer is in R3.



```

File Edit Tabs Help
[section .data
val1: .byte -60
val2: .byte 11
val3: .byte 16
.section .text
.globl _start
_start:
    ldr r0,=val1 @load into r0 val1 as an signed byte
    ldrsb r0, [r0]

    ldr r1,=val3 @load into r1 val3 as an unsigned byte
    ldrb r1, [r1]

    sub r1,r1,r0 @16 - (-60) = 76. R1 should contain 76

    ldr r3, =val2 @load into r3 val2 as a unsigned byte
    ldrb r3, [r3]

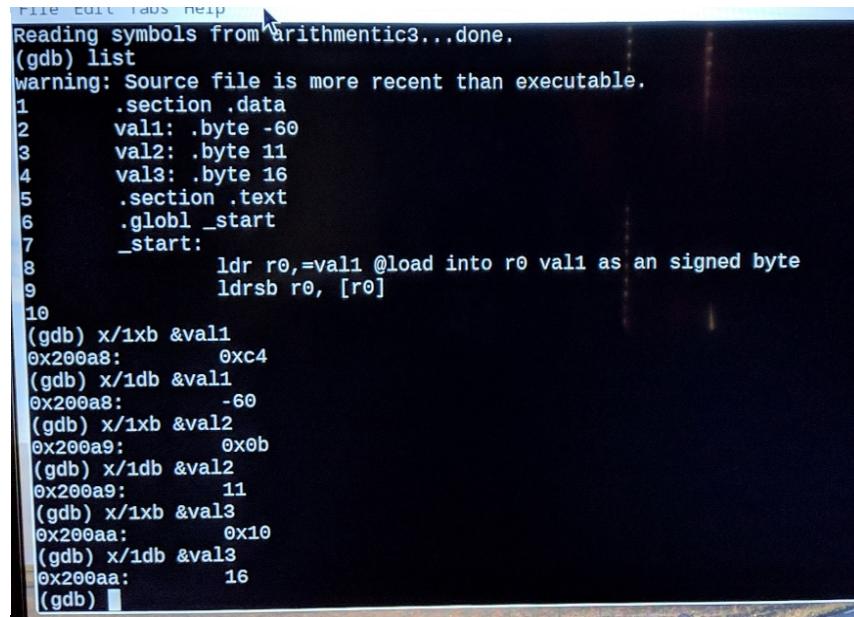
    add r3,r3,#3 @11 + 3 equals 14. R3 should contain 14

    add r3, r3,r1 @14 + 76 = 90. R3 should contain 90

    mov r7, #1
.end
-
"arithmentic3.s" 24 lines, 481 characters

```

I’ve assembled and linked the file and opened it in gdb. Upon entering gdb we used “list” to show the variables. Using the “x” command, we have shown the hexadecimal contents for each variable. We’ve also shown it as a decimal to fully confirm it because we did not want to do hexadecimal to decimal calculations.

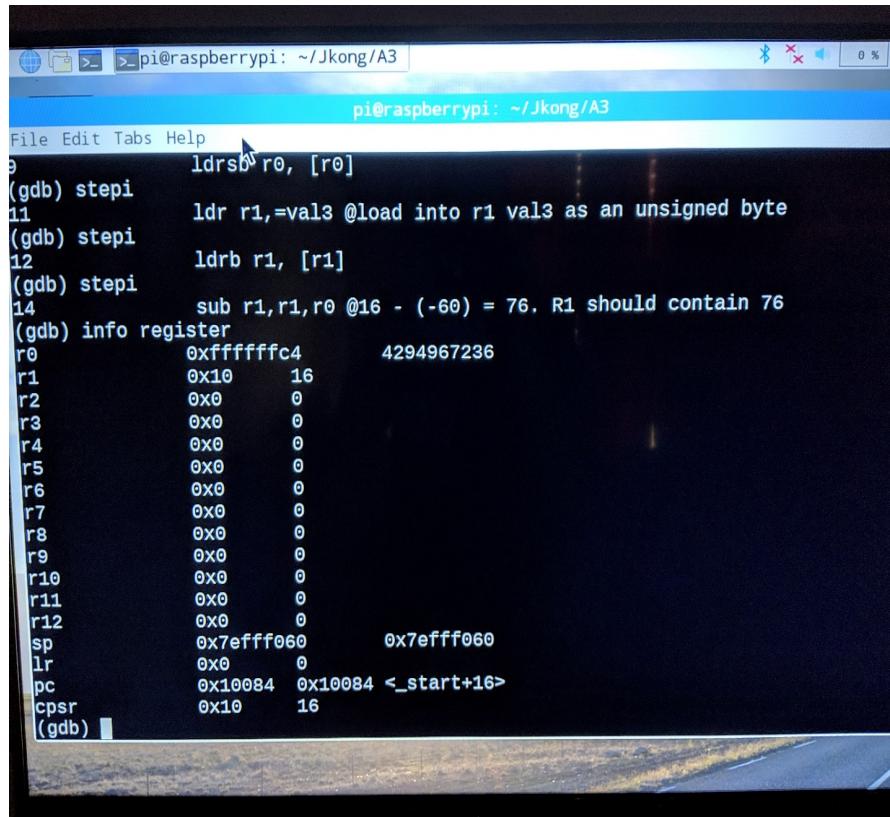


```

FILE EDIT TABS HELP
Reading symbols from arithmentic3...done.
(gdb) list
warning: Source file is more recent than executable.
1     .section .data
2     val1: .byte -60
3     val2: .byte 11
4     val3: .byte 16
5     .section .text
6     .globl _start
7     _start:
8         ldr r0,=val1 @load into r0 val1 as an signed byte
9             ldrsb r0, [r0]
10
(gdb) x/1xb &val1
0x200a8:          0xc4
(gdb) x/1db &val1
0x200a8:          -60
(gdb) x/1xb &val2
0x200a9:          0xeb
(gdb) x/1db &val2
0x200a9:          11
(gdb) x/1xb &val3
0x200aa:          0x10
(gdb) x/1db &val3
0x200aa:          16
(gdb)

```

We set a breakpoint before running the program and used “info register” to show the contents of the register as the program progresses. Here, it can be seen that var1 and var3 being loaded into R0 and R1 respectively.

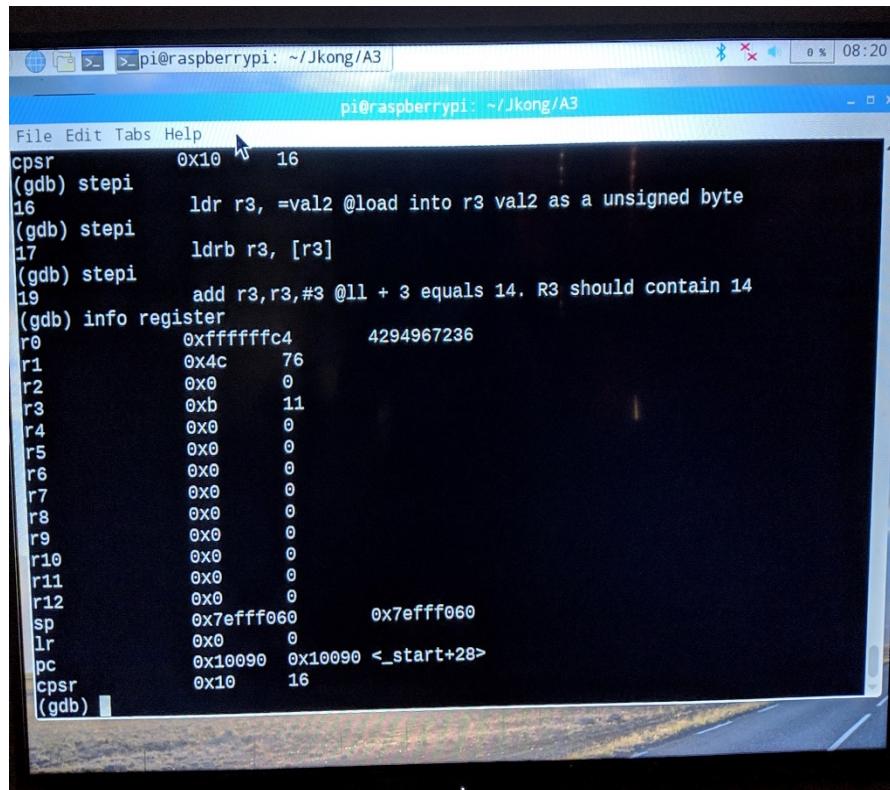


```

pi@raspberrypi: ~/Jkong/A3
File Edit Tabs Help
9      ldr r0, [r0]
(gdb) stepi
11     ldr r1,=val3 @load into r1 val3 as an unsigned byte
(gdb) stepi
12     ldrb r1, [r1]
(gdb) stepi
14     sub r1,r1,r0 @16 - (-60) = 76. R1 should contain 76
(gdb) info register
r0      0xfffffc4      4294967236
r1      0x10      16
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      0x10084      0x10084 <_start+16>
cpsr    0x10      16
(gdb)

```

Now, it can be seen that the results of R0 being subtracted from R1. The result is 76. Also, var2 has successfully been loaded into register R3.



```

pi@raspberrypi: ~/Jkong/A3
File Edit Tabs Help
cpsr    0x10      16
(gdb) stepi
16     ldr r3, =val2 @load into r3 val2 as a unsigned byte
(gdb) stepi
17     ldrb r3, [r3]
(gdb) stepi
19     add r3,r3,#3 @ll + 3 equals 14. R3 should contain 14
(gdb) info register
r0      0xfffffc4      4294967236
r1      0x4c      76
r2      0x0      0
r3      0xb      11
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      0x10090      0x10090 <_start+28>
cpsr    0x10      16
(gdb)

```

```

pi@raspberrypi: ~/Jkong/A3
File Edit Tabs Help
r12      0x0      0
sp       0x7efff060    0x7efff060
lr       0x0      0
pc       0x10090  0x10090 <_start+28>
cpsr     0x10      16
(gdb) stepi
21      add r3, r3,r1 @14 + 76 = 90. R3 should contain 90
(gdb) info register
r0      0xfffffc4    4294967236
r1      0x4c      76
r2      0x0      0
r3      0xe      14
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       0x10094  0x10094 <_start+32>
cpsr     0x10      16
(gdb)

```

Finally, just add R1 to R3 and the final result is 90. It can be seen that the CPSR register below with contents 0x10 which is 16 in decimal. This means that the contents of CPSR is as follows:

0000.0000.0001.0000

```

pi@raspberrypi: ~/Jkong/A3
File Edit Tabs Help
r12      0x0      0
sp       0x7efff060    0x7efff060
lr       0x0      0
pc       0x10094  0x10094 <_start+32>
cpsr     0x10      16
(gdb) stepi
23      mov r7, #1
(gdb) info register
r0      0xfffffc4    4294967236
r1      0x4c      76
r2      0x0      0
r3      0x5a      90
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       0x10098  0x10098 <_start+36>
cpsr     0x10      16
(gdb)

```

In the CPSR register, the Sign (Negative) Flag is the 32nd bit and it is not set at the end of the program because the answer is positive.

APPENDIX:

Slack

<https://hacks-org.slack.com/messages/CFTR0NGG6/team/UFSJLJGSH/>

The screenshot shows a Slack interface with the URL <https://hacks-org.slack.com/messages/CFTR0NGG6/team/UFSJLJGSH/> in the address bar. A modal window at the top says "Slack needs your permission to enable desktop notifications." Below it, the #general channel has 6 messages and 0 attachments. The messages are:

- Jeffery Kong** 9:23 AM: test scores are posted if you did not know
- Mohamed** 11:42 AM: I uploaded task3 draft If you guys want to edit go for it <https://github.com/Team-hacks/assignment3>
- GitHub** 11:43 AM: Contribute to Team-hacks/assignment3 development by creating an account on GitHub.
- here is the youtube link** <https://youtu.be/F0U5qWkhyuc>
- YouTube | Team Hacks Assignment 3**: A video thumbnail showing four people sitting around a table.
- Vincent** 11:59 AM: Just fyi, [@Jeffery Kong](#), You should use .sh for a signed halfword. I have the Pi but if I pass it off to [@Sabina Maharjan](#) today, she'll need to take the screenshot of the updated code for the report

At the bottom, there's a message input field with "+ Message #general" and a send button.

GitHub

<https://github.com/Team-hacks/assignment3/blob/master/assignment3.docx>

The screenshot shows the GitHub profile for the user "Team-hacks". The profile page includes a header with navigation icons and links to GitHub, Inc. [US] and the user's profile URL. Below the header are sections for Overview, Repositories (3), Projects (0), Stars (0), Followers (0), and Following (0). A large pink bar chart is displayed on the left side of the profile page.

Popular repositories:

- hacks**: 1 commit
- assignment2**
- assignment3**

Contribution activity:

14 contributions in the last year

A heatmap showing contributions by month and day. Contributions are represented by dark green squares. The heatmap shows activity primarily in January and December of the previous year.

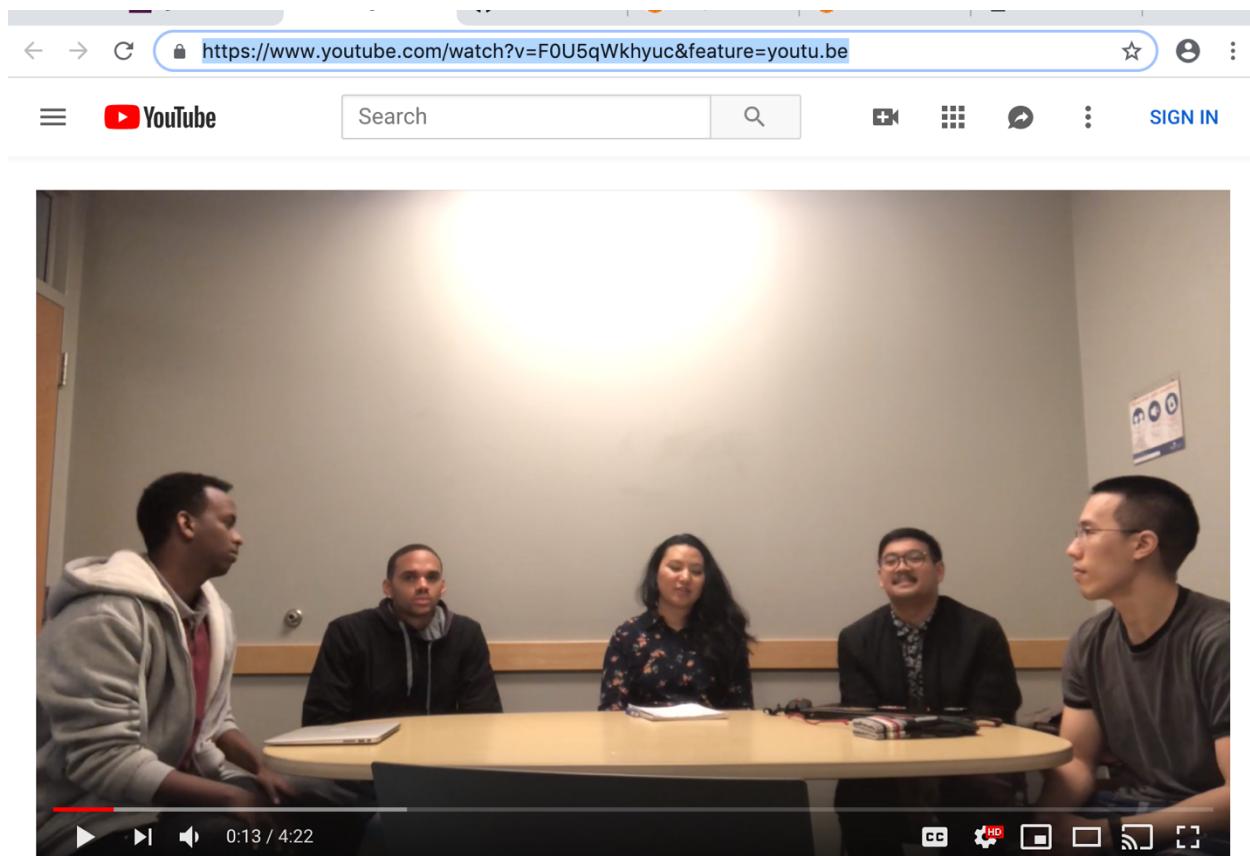
Contribution activity (March 2019):

- Created 1 commit in 1 repository: **Team-hacks/assignment3** 1 commit
- Created 1 repository: **Team-hacks/assignment3** Mar 6

Show more activity

You-Tube:

<https://www.youtube.com/watch?v=F0U5qWkhyuc&feature=youtu.be>



Assignment 3

Unlisted

2 views

0 likes | 0 dislikes | SHARE | SAVE | ...



Team Hacks

Uploaded on Mar 5, 2019

SUBSCRIBE 0

Category

People & Blogs



THE MOST RECKLESS PILOTS. LOWEST FLIGHTS YOU'VE NEVER SEEN BEFORE

#Mind Warehouse