

**Gcc trapnotThe Assemblers
Spring 2019**

Developing Soft and Parallel Programming Skills Using Project-Based Learning

**Authors:
Aaja Christie
Davidson Fleurantin
Mamadou Diallo
Sheng Chen
Matthew Kabat**

Planning and Scheduling

Assignee Name	Email	Task	Duration (hours)	Dependenc y	Due Date	Note
---------------	-------	------	---------------------	----------------	----------	------

	(@student.gsu.edu)				(March 8th)	
Aaja Christie (Coordinator)	achristie3	Task 2 Task 4	5	Github Slack Raspberry pi 3 B+	March 7th	
Davidson Fleurantin	dfleurantin1	Parallel Programming	3	Raspberry pi 3 B+	March 4th	
Mamadou Diallo	mdiallo15	Task 5 Report	2	Google Docs	March 7	
Sheng Chen	schen36	Task 6 Video editing	2	Adobe Premiere Pro CC 2018	March 7	
Matthew Kabat	mkabat1	Task 3A	2		March 7th	

Task 3: Programming Skills

Foundations

1. Define the following:

- Task - a discrete program or set of instructions that is run by the processor. Multiple tasks are done by multiple processors in a parallel program.
- Pipelining - the process of breaking down a task into discrete chunks that can each be done by a different processing unit
- Shared Memory -
 - Hardware View: All processors have direct access to common physical memory.
 - Software View: All tasks have same view of memory and can access memory regardless of its physical location.
- Communications - exchanging data to do parallel computing
- Synchronization - the coordination of parallel tasks in real time. Often implemented by establishing synchronization points in an application.

2. Classify parallel computers based on Flynn's taxonomy. Briefly describe every one of them.

- Single Instruction, Single Data (SISD): a single processing unit performs one instruction upon one piece of data at a time.
- Single Instruction, Multiple Data (SIMD): multiple processing units perform the same instruction upon different pieces of data.
- Multiple Instruction, Single Data (MISD): multiple processing units perform different instructions upon a single piece of data.
- Multiple Instruction, Multiple Data (MIMD): multiple processing units perform different instructions upon different pieces of data.

3. What are the Parallel Programming Models?

Parallel programming models are different ways to implement parallel computing. Examples include Shared Memory, Distributed Memory, and Threads.

4. List and briefly describe the types of Parallel Computer Memory Architectures. What type is used by OpenMP and why?

- Shared Memory Model (without threads):
 - processes/tasks share a common address space, which they read and write to asynchronously
 - Various mechanisms such as locks / semaphores are used to control access to the shared memory, resolve contentions and to prevent race conditions and deadlocks.
- Shared Memory Model (with threads):

- A single "heavy weight" process has multiple "light weight", concurrent execution paths called threads
- Each thread has local data, but also, shares the entire resources of the main program
- OpenMP uses a threaded Shared Memory model. This allows for greater portability and increased ease of use for the programmer.

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

The main difference is that an unthreaded Shared Memory Model is far more intensive to setup and manage. However, it does make communication of data easier because data is not “owned” in this model.

6. What is Parallel Programming? (in your own words) -

Parallel programming is when you have multiple processing units acting in a single machine or cluster to solve a problem.

7. What is system on chip (SoC)? Does Raspberry PI use system on SoC? -

A System-on-a-Chip integrates several discrete features into a single chip. They usually contain a CPU, GPU, memory, a USB controller, power management circuits, and wireless radios. The Raspberry Pi uses a SoC which contains a CPU, GPU, and RAM.

8. Explain what the advantages are of having a System on a Chip rather than separate CPU, GPU and RAM components.

For only a slight increase in size over a CPU, a SOC has far more functionality. Because various components are integrated, the absolute minimum size of a device using a SOC is greatly reduced when compared to a device with a discrete CPU. In addition, SOC's use significantly less power due to the components being physically closer and thus requiring less wiring. Finally, building a computer using SOC's is cheaper because fewer discrete physical chips are necessary.

Parallel Programming

Code 1 image. pLoop: Illustration of OpenMP's default parallel for loop

```
#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;
}
```

pLoop/4. Result of pLoop with 4 threads.

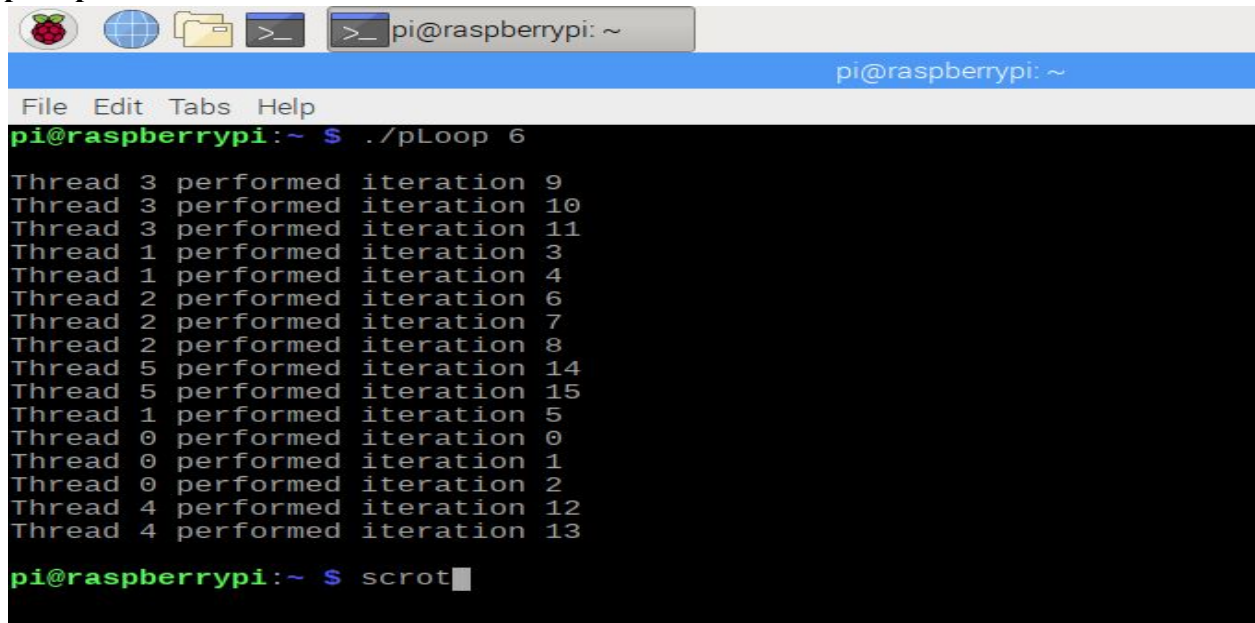
```
pi@raspberrypi: ~
File Edit Tabs Help
pi@raspberrypi:~ $ nano parallelLoopEqualChunks.c
pi@raspberrypi:~ $ gcc parallelLoopEqualChunks.c -o pLoop -fopenmp
pi@raspberrypi:~ $ ./pLoop 4

Thread 3 performed iteration 12
Thread 3 performed iteration 13
Thread 3 performed iteration 14
Thread 3 performed iteration 15
Thread 1 performed iteration 4
Thread 1 performed iteration 5
Thread 1 performed iteration 6
Thread 1 performed iteration 7
Thread 2 performed iteration 8
Thread 2 performed iteration 9
Thread 2 performed iteration 10
Thread 2 performed iteration 11
Thread 0 performed iteration 0
Thread 0 performed iteration 1
Thread 0 performed iteration 2
Thread 0 performed iteration 3

pi@raspberrypi:~ $ scrot
```

This is the result when pLoop ran on 4 threads. The loop has 16 iterations, and each thread will perform 4 iterations. The OpenMP pragma is applied in front of the loop.

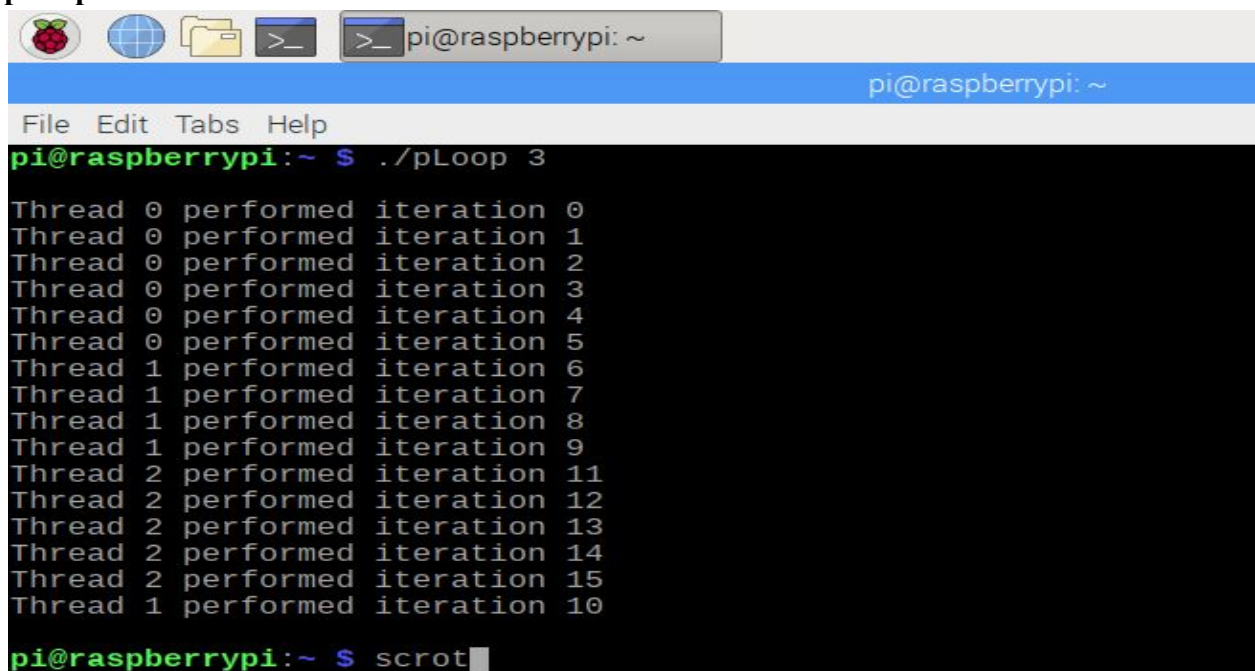
pLoop/6. Code ran on 6 threads



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

When the code ran on 6 threads, some threads performed 2 iterations, others performed 3 iterations. The first 4 threads (0-3) has 3 iterations, the last 2 (4,5) has two iterations.

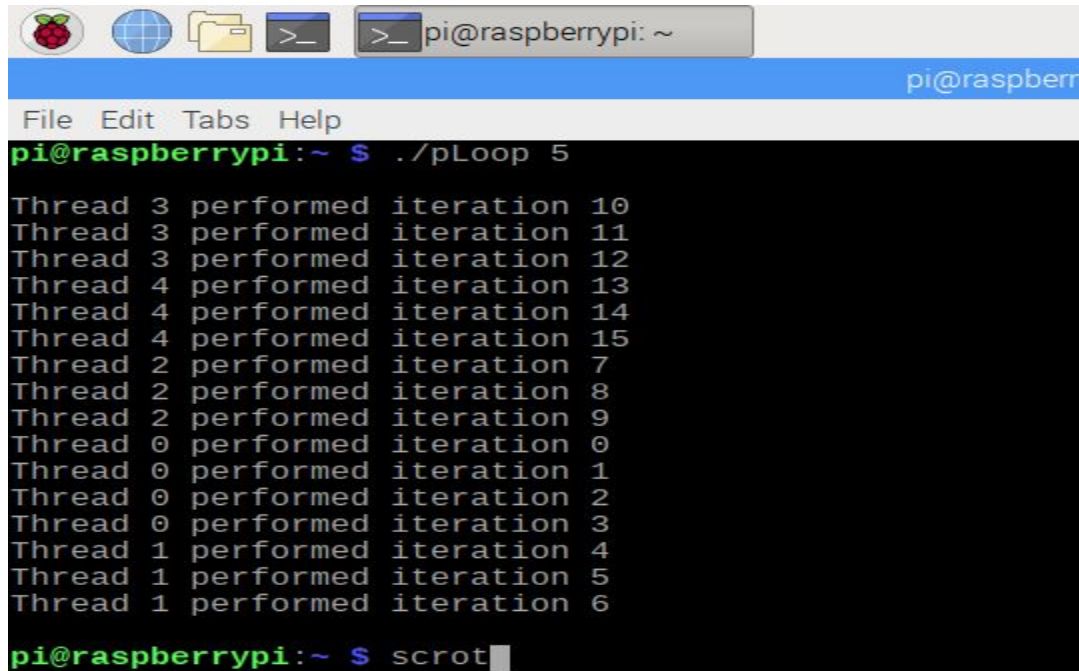
pLoop/3



```
pi@raspberrypi:~ $ ./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 1 performed iteration 6
Thread 1 performed iteration 7
Thread 1 performed iteration 8
Thread 1 performed iteration 9
Thread 1 performed iteration 10
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
pi@raspberrypi:~ $ scrot
```

With the number of threads not divisible by 16, the first thread ran 6 iterations, threads 1 and 2 ran 5 iterations each. The first thread runs the most iterations, then it goes down from there.

pLoop/5



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

With 5 threads running the program, the first thread (0) runs the most iterations (4), then the remaining threads run 3 iterations each.

Code 2. Loop iteration in chunks of size 1 using OpenMP

```

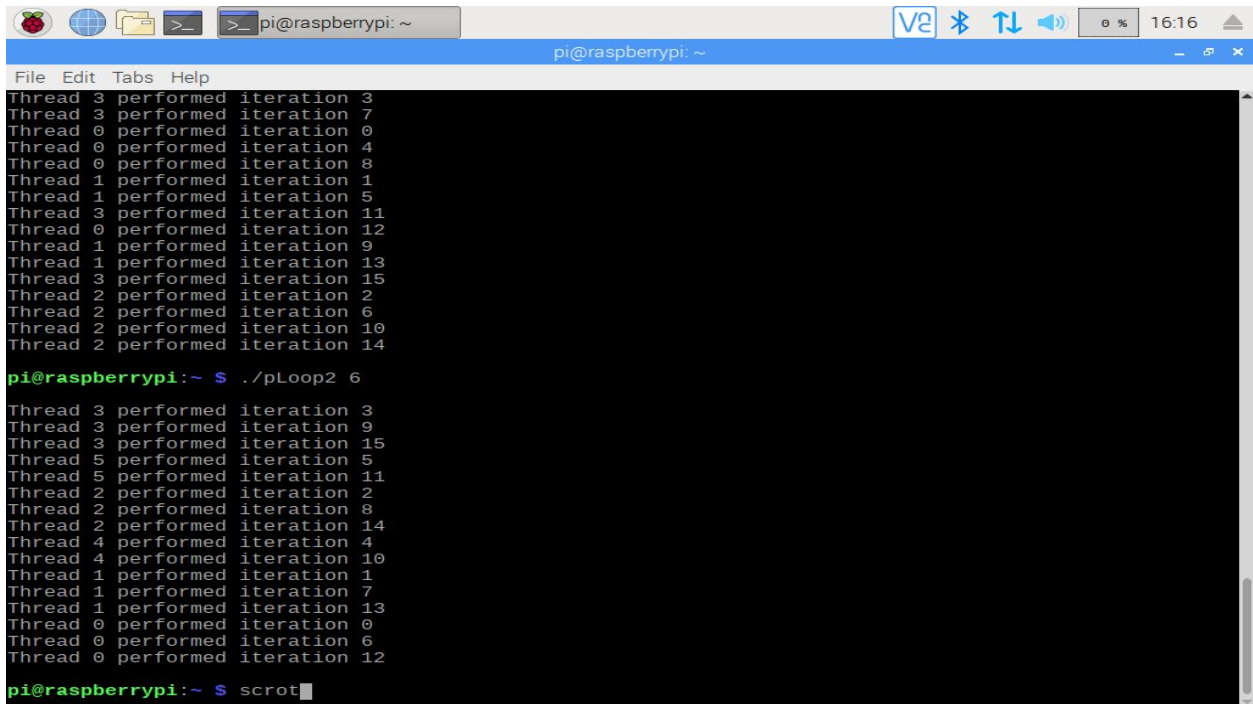
#include <stdio.h>
#include <omp.h>
#include <stdlib.h>

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

```

This code shows how OpenMP can use the first thread to do one iteration, the next one does the second iteration and so on until the last thread, when each thread does its iteration, the process starts again with the first thread. Each thread performs the same of work, this is called static scheduling.

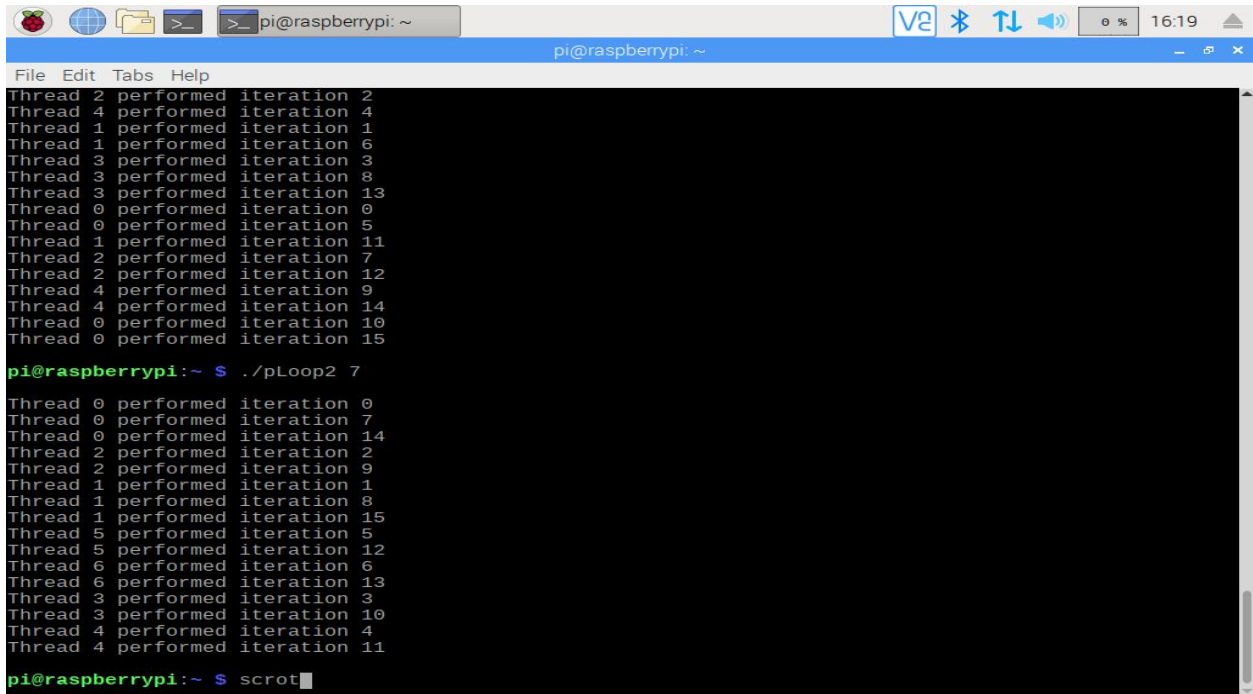
pLoop2_4&6. The following image uses 4 and 6 threads



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

Static scheduling with 4 and 6 threads, the iterations on the top uses 4 threads. Thread 0 does iteration 0, thread 1 does iteration 1 and so on, then the process starts again with thread 0 doing iteration 4 and so on. The same is true using 6 threads.

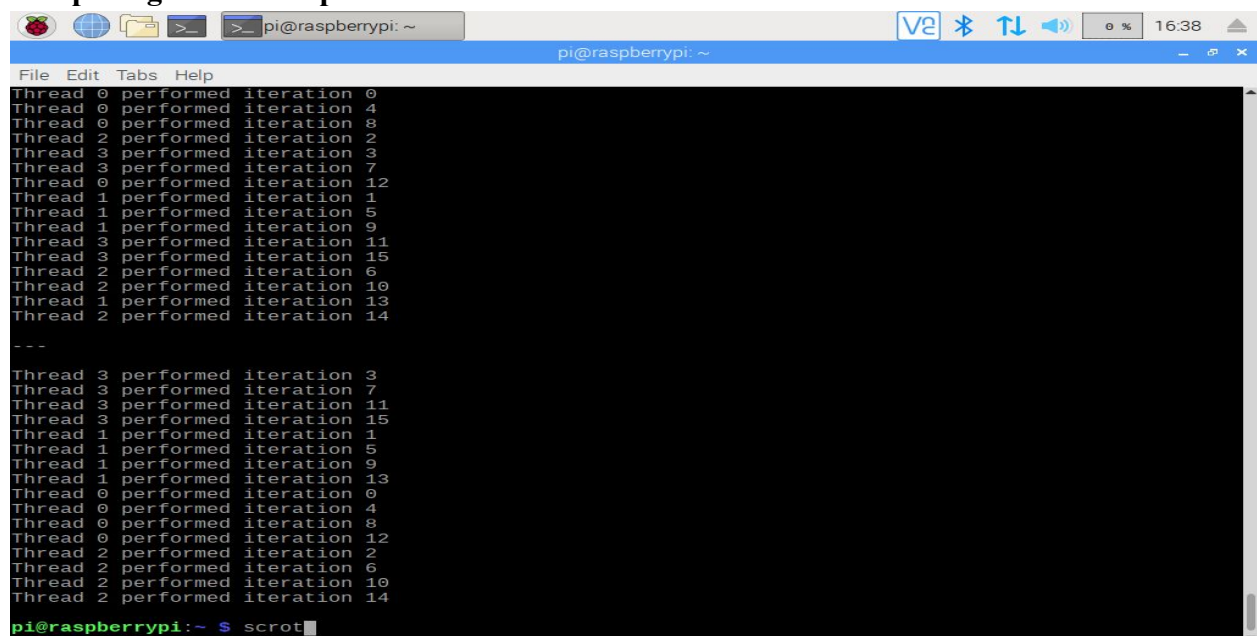
pLoop2_3&5. Static scheduling with 5 and 7 threads



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

The top part shows the 16 iterations using 5 threads, as expected thread 0 does 4 iterations, the remaining threads do 3 each, as shown before in the previous images. Static scheduling is in works here.

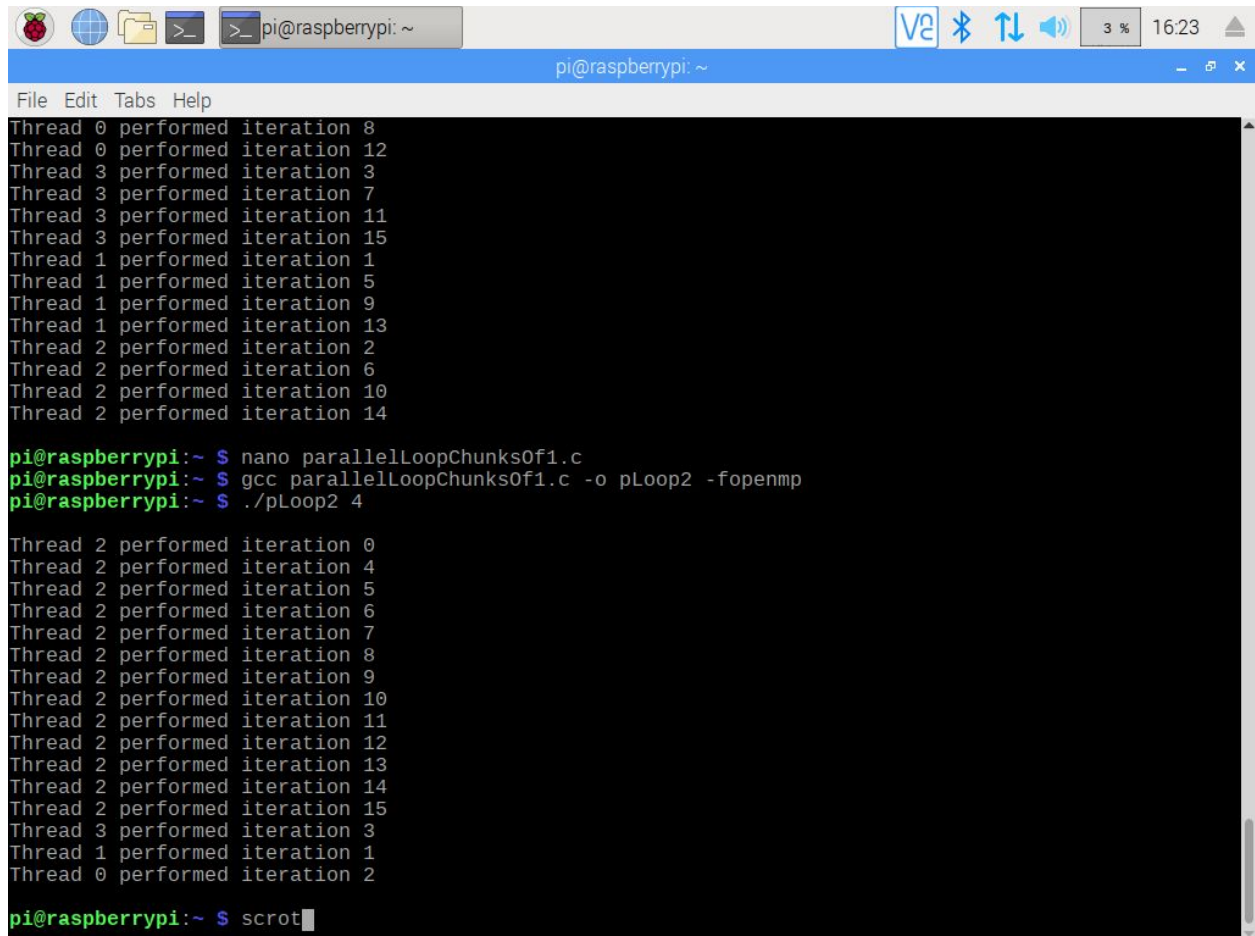
Comparing the two loops inside the code



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

When the comments were removed on the second loop, both loops were run, the first loop runs statically, the second one using OpenMP's pragma parallel. Both loops performed the same static operations, they have exactly the same output. Comparing the two loops with the 'equalChunks' loop, the equal chunk loop runs the iteration in succession, meaning, thread 0 performs iteration 0,1,2,3, thread 1 performs iterations 4,5,6,7, and so on. Here, the operations are performed statically.

Dynamic scheduling



```
pi@raspberrypi: ~  
File Edit Tabs Help  
Thread 0 performed iteration 8  
Thread 0 performed iteration 12  
Thread 3 performed iteration 3  
Thread 3 performed iteration 7  
Thread 3 performed iteration 11  
Thread 3 performed iteration 15  
Thread 1 performed iteration 1  
Thread 1 performed iteration 5  
Thread 1 performed iteration 9  
Thread 1 performed iteration 13  
Thread 2 performed iteration 2  
Thread 2 performed iteration 6  
Thread 2 performed iteration 10  
Thread 2 performed iteration 14  
  
pi@raspberrypi:~ $ nano parallelLoopChunksOf1.c  
pi@raspberrypi:~ $ gcc parallelLoopChunksOf1.c -o pLoop2 -fopenmp  
pi@raspberrypi:~ $ ./pLoop2 4  
  
Thread 2 performed iteration 0  
Thread 2 performed iteration 4  
Thread 2 performed iteration 5  
Thread 2 performed iteration 6  
Thread 2 performed iteration 7  
Thread 2 performed iteration 8  
Thread 2 performed iteration 9  
Thread 2 performed iteration 10  
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 3 performed iteration 3  
Thread 1 performed iteration 1  
Thread 0 performed iteration 2  
  
pi@raspberrypi:~ $ scrot
```

This image shows the difference between static and dynamic scheduling. We have seen how static scheduling works (top part). The bottom part uses 4 threads and the code runs dynamically. As seen, thread 2 performs most of the iterations, in this case, 13 iterations. In dynamic scheduling, the threads are not doing the same amount of work.

Dependencies in Loops

```

#include <stdio.h> // printf()
#include <omp.h> // OpenMP
#include <stdlib.h> // rand()

void initialize(int* a, int n);
int sequentialSum(int* a, int n);
int parallelSum(int* a, int n);

#define SIZE 1000000

int main(int argc, char** argv){
int array[SIZE];

if(argc>1){
    omp_set_num_threads(atoi(argv[1]));
}

initialize(array, SIZE);
printf("\nSequential sum: %d\nParallel sum: %d\n",
    sequentialSum(array, SIZE),
    parallelSum(array, SIZE));

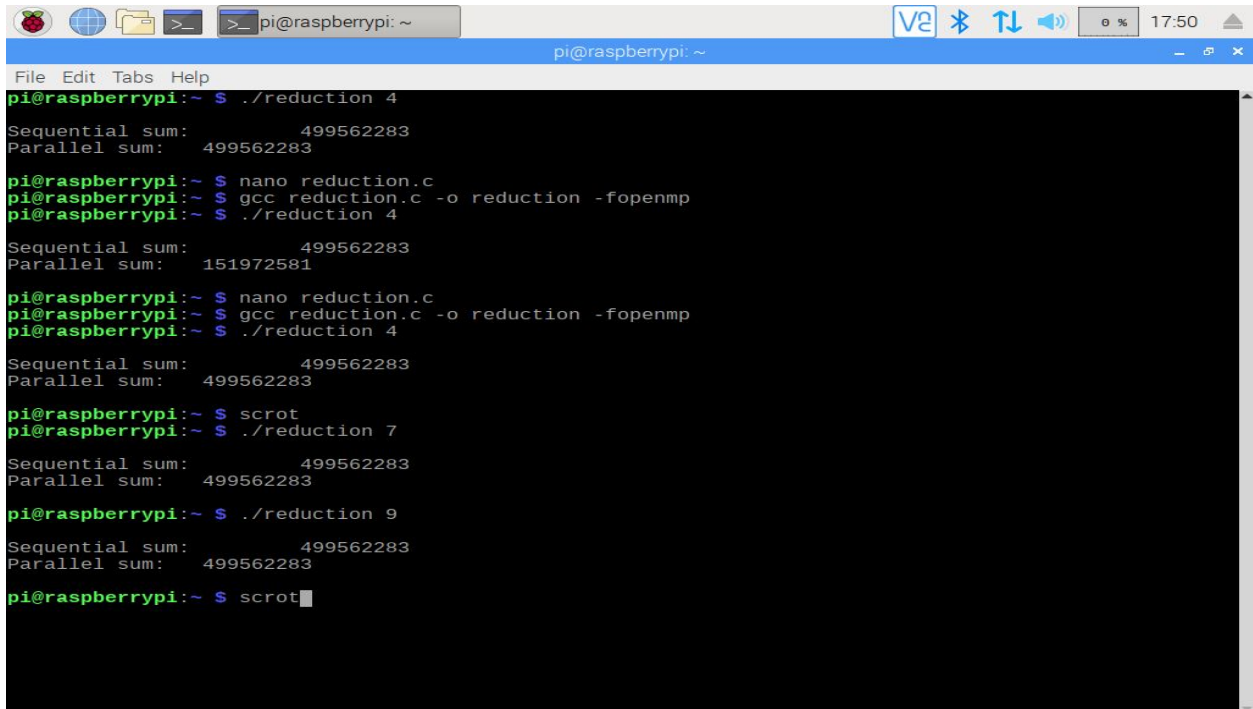
return 0;
}
/* fill array with random values */
void initialize(int* a, int n){
    int i;
    for(i=0; i<n; i++){
        a[i] = rand()%1000;
    }
}

/* sum the array sequentially */
int sequentialSum(int* a, int n){
    int sum=0;
    int i;
    for(i=0; i<n; i++){
        sum +=a[i];
    }
    return sum;
}

/* sum the array using multiple threads */
int parallelSum(int* a, int n){
    int sum=0;
    int i;
    #pragma omp parallel for reduction(+:sum)
    for(i=0; i<n; i++){
        sum +=a[i];
    }
    return sum;
}

```

This code shows how to sequentially use the sum accumulator and its parallel computation implementation using OpenMP.

A terminal window on a Raspberry Pi showing the execution of a program with OpenMP reduction. The user runs './reduction 4', './reduction 7', and './reduction 9'. Each time, the sequential sum is 499562283. For 4 threads, the parallel sum is 151972581 (incorrect). For 7 and 9 threads, the parallel sum is 499562283 (correct). The user also runs 'nano reduction.c', 'gcc reduction.c -o reduction -fopenmp', and 'scrot' to capture the output.

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

pi@raspberrypi:~ $ nano reduction.c
pi@raspberrypi:~ $ gcc reduction.c -o reduction -fopenmp
pi@raspberrypi:~ $ ./reduction 4
Sequential sum: 499562283
Parallel sum: 151972581

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

pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ ./reduction 7
Sequential sum: 499562283
Parallel sum: 499562283

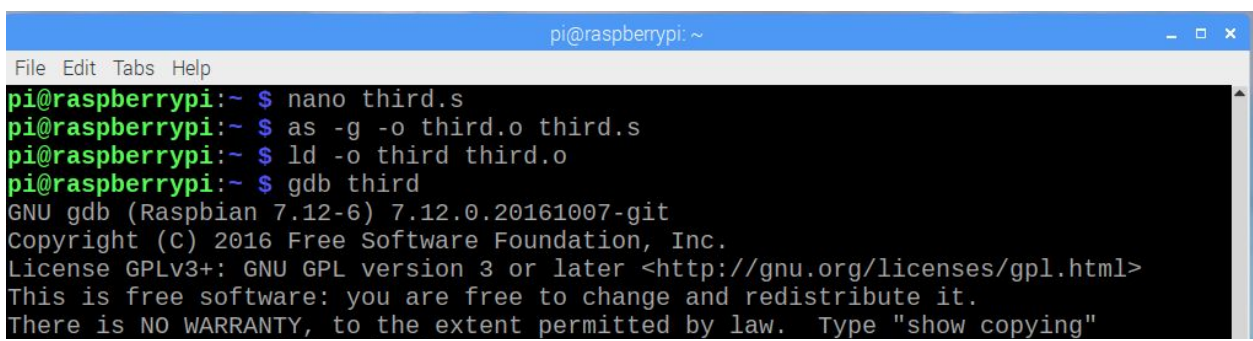
pi@raspberrypi:~ $ ./reduction 9
Sequential sum: 499562283
Parallel sum: 499562283

pi@raspberrypi:~ $ scrot
```

Both sequentialSum and ParallelSum results in the same output when when #pragma omp was commented out. Both have an amount of 499562283. When the #pragma omp directive was uncommented, the Parallel sum produces the wrong output 151972581. When the reduction (+:sum) was uncommented, along with the #pragma directive, the parallel method goes back to the correct sum. That shows that the sum accumulator depends on what all the threads. All threads performed their own work and sum up their work, when they are all done, the reduction clause allows the system to compute a final sum of each of the thread's sum. Using 7 and 9 threads, the output is always the same for both sequential and parallel summation.

Task 4

Part 1

A terminal window on a Raspberry Pi showing the compilation of a file named 'third.s'. The user runs 'nano third.s', 'as -g -o third.o third.s', 'ld -o third third.o', and 'gdb third'. The terminal shows the GNU gdb version 7.12-6 and its license information.

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

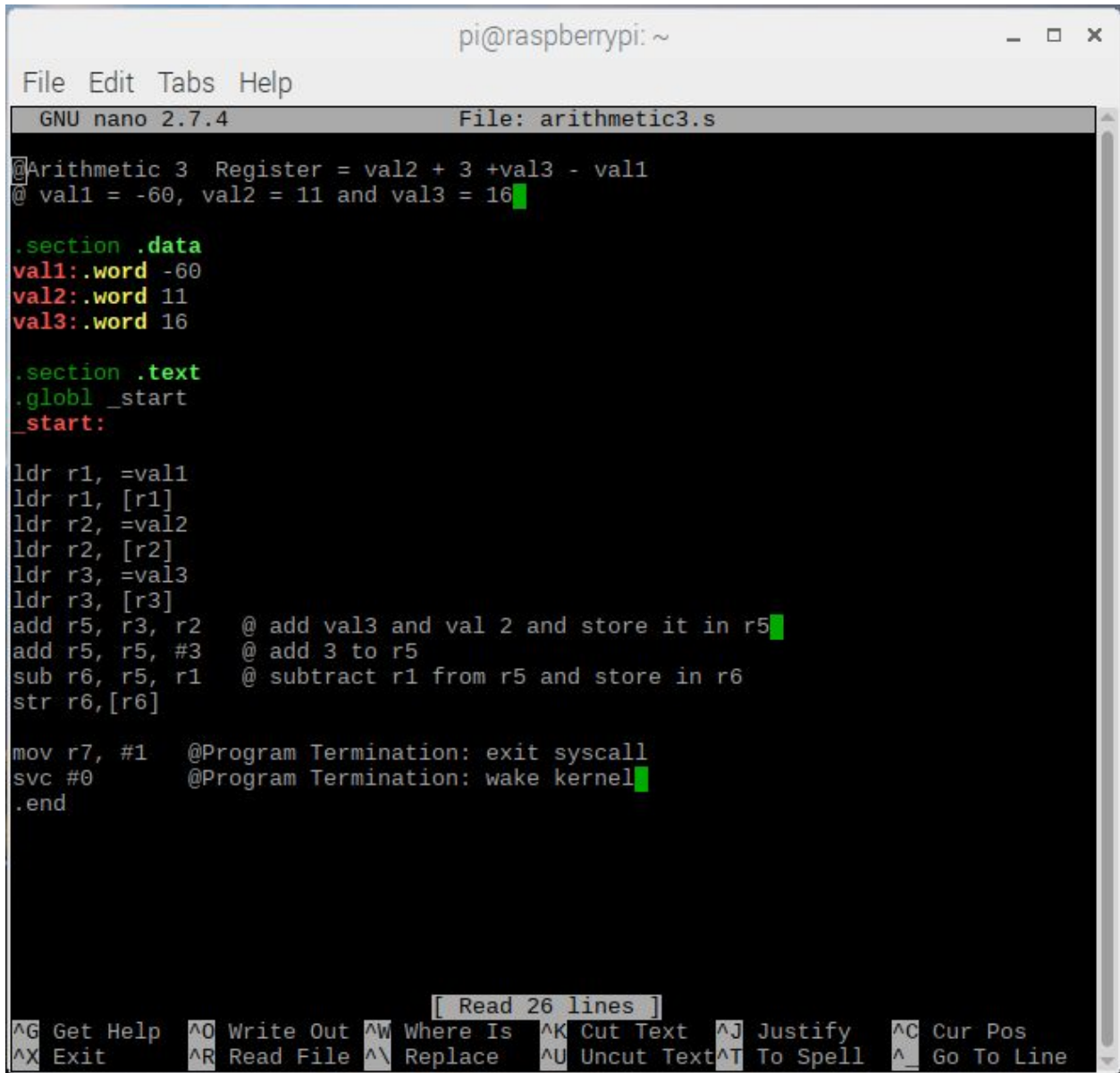

This is assembling, linking, and running with a breakpoint in the GNU Debugger after the code has been fixed. The issue it had was with declaring the signed halfword. For some reason ARM won't let us declare a signed halfword (or normal halfwords for that matter) but I noticed when you don't declare it as signed ARM still recognizes it as signed so i just declared it as an unsigned word and it worked.

```
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 3
1          @Third program
2      .section .data
3      a: .word -2 @16-bit signed integer
4
5      .section .text
6      .globl _start
7      _start:
8
9      @ the following code attempts to load va lues into a set of registers an
d
10     @ needs debugging
(gdb) b 3
Breakpoint 1 at 0x10078: file third.s, line 3.
(gdb) x/1xh 0x10078
0x10078 <_start+4>:      0x1000
(gdb) x/1xsh 0x10078
0x10078 <_start+4>:      u"1[0]x20ff[0]x3101[0] [0] [0]
(gdb) stepi
The program is not being run.
(gdb) run
Starting program: /home/pi/third

Breakpoint 1, _start () at third.s:13
13      mov r1, #0xFFFFFFFF@ = -1(signed)
```

While displaying a halfword in hexadecimal, this is what happened when I ran it with and without the s. With the s, there were some weird characters displayed that I don't recognize.

Part 2



```
pi@raspberrypi: ~
File Edit Tabs Help
GNU nano 2.7.4 File: arithmetic3.s
@Arithmetic 3 Register = val2 + 3 +val3 - val1
@ val1 = -60, val2 = 11 and val3 = 16

.section .data
val1:.word -60
val2:.word 11
val3:.word 16

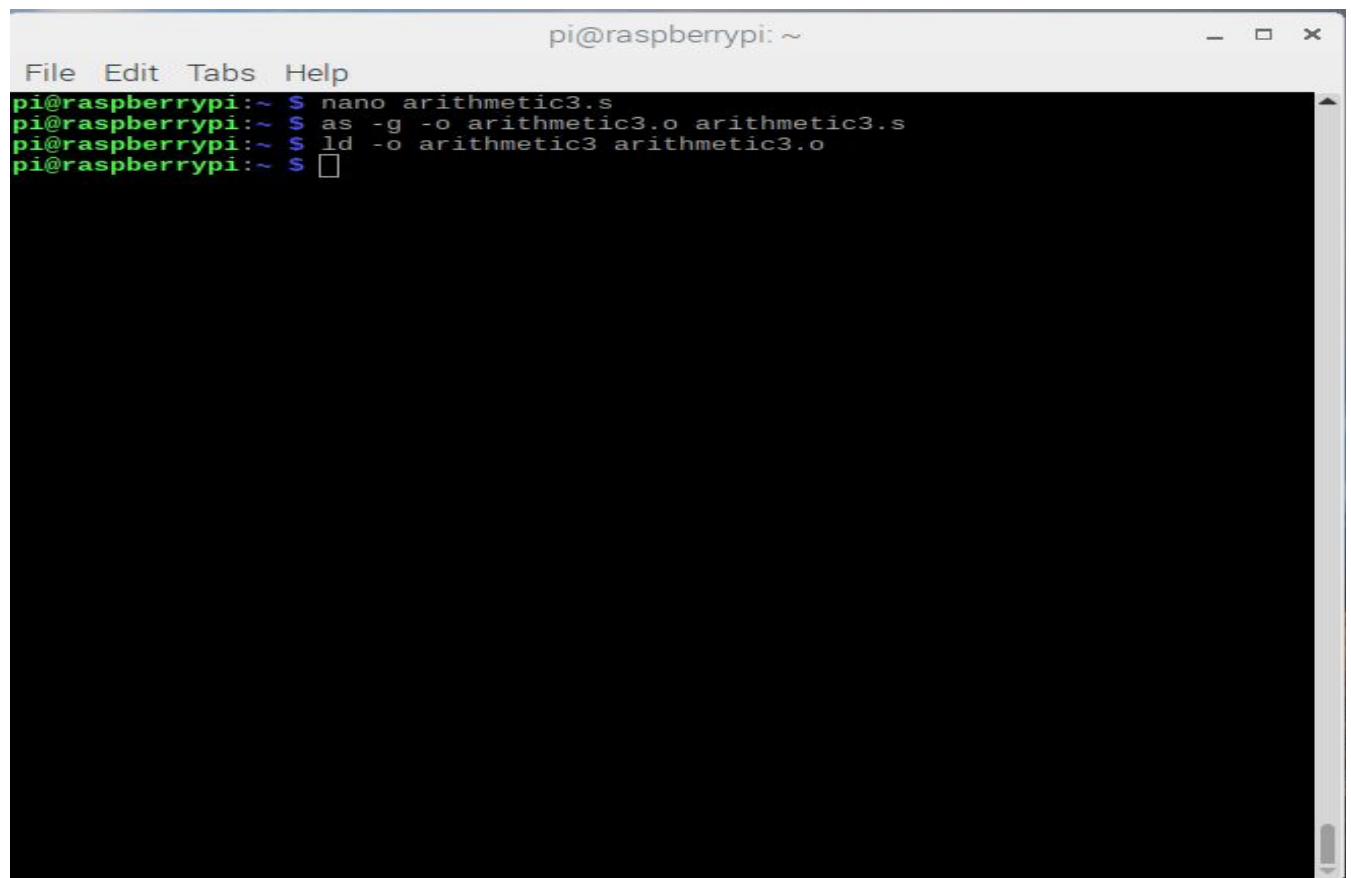
.section .text
.globl _start
_start:

ldr r1, =val1
ldr r1, [r1]
ldr r2, =val2
ldr r2, [r2]
ldr r3, =val3
ldr r3, [r3]
add r5, r3, r2 @ add val3 and val 2 and store it in r5
add r5, r5, #3 @ add 3 to r5
sub r6, r5, r1 @ subtract r1 from r5 and store in r6
str r6, [r6]

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

[ Read 26 lines ]
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Spell ^_ Go To Line
```

This is our code for calculating $\text{val2} + 3 + \text{val3} - \text{val1}$.

A terminal window titled "pi@raspberrypi: ~" with a menu bar containing "File", "Edit", "Tabs", and "Help". The terminal shows a sequence of commands: "nano arithmetic3.s", "as -g -o arithmetic3.o arithmetic3.s", and "ld -o arithmetic3 arithmetic3.o". The prompt "pi@raspberrypi:~" is repeated for each command. The last line shows the prompt followed by a cursor.

```
pi@raspberrypi:~ $ nano arithmetic3.s
pi@raspberrypi:~ $ as -g -o arithmetic3.o arithmetic3.s
pi@raspberrypi:~ $ ld -o arithmetic3 arithmetic3.o
pi@raspberrypi:~ $
```

This is assembling and linking the code.


```
pi@raspberrypi: ~  
File Edit Tabs Help  
pi@raspberrypi:~ $ ld -o arithmetic3 arithmetic3.o  
pi@raspberrypi:~ $ gdb arithmetic3  
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 arithmetic3...done.  
(gdb) list 17  
12  
13     ldr r1, =val1  
14     ldr r1, [r1]  
15     ldr r2, =val2  
16     ldr r2, [r2]  
17     ldr r3, =val3  
18     ldr r3, [r3]  
19     add r5, r3, r2    @ add val3 and val 2 and store it in r5  
20     add r5, r5, #3    @ add 3 to r5  
21     sub r6, r5, r1    @ subtract r1 from r5 and store in r6  
(gdb) gdb b 10  
Undefined command: "gdb". Try "help".  
(gdb) b 10  
Breakpoint 1 at 0x10078: file arithmetic3.s, line 10.  
(gdb) run  
Starting program: /home/pi/arithmetic3  
  
Breakpoint 1, _start () at arithmetic3.s:14  
14     ldr r1, [r1]  
(gdb) □
```

This is opening up the GNU Debugger, adding a breakpoint and running the program.

```
pi@raspberrypi: ~  
File Edit Tabs Help  
r0          0x0      0  
r1          0xffffffffc4    4294967236  
r2          0xb      11  
r3          0x200b8    131256  
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          0x7efff070    0x7efff070  
lr          0x0      0  
pc          0x10088    0x10088 <_start+20>  
cpsr        0x10      16  
(gdb) stepi  
19      add r5, r3, r2    @ add val3 and val 2 and store it in r5  
(gdb) info registers  
r0          0x0      0  
r1          0xffffffffc4    4294967236  
r2          0xb      11  
r3          0x10      16  
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          0x7efff070    0x7efff070  
lr          0x0      0  
pc          0x1008c    0x1008c <_start+24>  
cpsr        0x10      16  
(gdb) ☐
```

Here are the info registers after all the variables had been assigned and loaded to registers.

```
pi@raspberrypi: ~  
File Edit Tabs Help  
r0          0x0      0  
r1          0xffffffffc4    4294967236  
r2          0xb      11  
r3          0x10     16  
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          0x7efff070    0x7efff070  
lr          0x0      0  
pc          0x1008c  0x1008c <_start+24>  
cpsr        0x10     16  
(gdb) stepi  
20      add r5, r5, #3    @ add 3 to r5  
(gdb) info registers  
r0          0x0      0  
r1          0xffffffffc4    4294967236  
r2          0xb      11  
r3          0x10     16  
r4          0x0      0  
r5          0x1b     27  
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          0x10090  0x10090 <_start+28>  
cpsr        0x10     16  
(gdb) 
```

This is the info registers after adding val2 and val3 together and storing that in r5.

```
pi@raspberrypi: ~  
File Edit Tabs Help  
r0          0x0      0  
r1          0xffffffffc4    4294967236  
r2          0xb      11  
r3          0x10     16  
r4          0x0      0  
r5          0x1b     27  
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          0x10090  0x10090 <_start+28>  
cpsr        0x10     16  
(gdb) stepi  
21      sub r6, r5, r1    @ subtract r1 from r5 and store in r6  
(gdb) info registers  
r0          0x0      0  
r1          0xffffffffc4    4294967236  
r2          0xb      11  
r3          0x10     16  
r4          0x0      0  
r5          0x1e     30  
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          0x10094  0x10094 <_start+32>  
cpsr        0x10     16  
(gdb) ☐
```

This is adding the immediate 3 to the sum of val2 and val3 which is stored in r5 and storing that in r5.

```
pi@raspberrypi: ~  
File Edit Tabs Help  
r0          0x0      0  
r1          0xffffffffc4    4294967236  
r2          0xb      11  
r3          0x10     16  
r4          0x0      0  
r5          0x1e     30  
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          0x10094  0x10094 <_start+32>  
cpsr        0x10     16  
(gdb) stepi  
22      str r6,[r6]  
(gdb) info registers  
r0          0x0      0  
r1          0xffffffffc4    4294967236  
r2          0xb      11  
r3          0x10     16  
r4          0x0      0  
r5          0x1e     30  
r6          0x5a     90  
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          0x10098  0x10098 <_start+36>  
cpsr        0x10     16  
(gdb) □
```

This is the registers after all the lines have been executed. I took the sum from register r5 and subtracted a negative 60 from it to get 90 which I then store in r6.

Appendix

Youtube Channel: [The Assemblers](#)

Assignment Video: <https://www.youtube.com/watch?v=qZHyvQ0xLJg>

Slack: <https://the-assemblers.slack.com/messages/CFSQ2GTDX/>

Github: <https://github.com/TheAssemblers>

The screenshot shows a GitHub repository page for 'TheAssemblers / CSC3210---The-Assemblers'. The repository has 0 Watchers, 0 Stars, and 0 Forks. The 'Projects' tab is selected, showing a Kanban board for 'CSC3210_ProjectA2' (updated 4 minutes ago). The board has two columns: 'In Progress' (2 tasks) and 'Done' (4 tasks). A third column is labeled '+ Add column'.

In Progress	Done
Task 6 (Editing Video) Added by TheAssemblers	Task 2 (Github and Slack) Added by TheAssemblers
Task 5 (Report) Added by TheAssemblers	Task 4 (ARM programming) Added by TheAssemblers
	Task 3 part B (Parallel Programming Basics) Added by TheAssemblers
	Task 3 part A (Parallel Programming Foundations: Questions) Added by TheAssemblers

The screenshot shows a Slack channel named '#general' with 6 members. The channel is part of a workspace named 'The Assemblers'. The messages are as follows:

- Mamadou** 12:30 PM: Hello Guys, My name is Mamadou Diallo. I am interested in Software Engineering, Artificial Intelligence and Data Science. My expectations from this group is to learn how to effectively manage my time and as well as the parallel aspect of the ARM architecture. My task for this assignment is creating the slack and video editing.
- Davidson Fleurantin** 1:26 PM: Hi, I am Davidson Fleurantin. I love bioinformatics, machine learning. To be a great programmer, you need to know how the computer hardware works. I hope to learn more about the hardware so I can become a great programmer. I expect this group to be efficient, to work well together to design great programs. I will be writing the report. The most challenging of this assignment is programming using the raspberry pi. With practice, I will be able to master this part.
- Sheng** 1:27 PM: My name is Sheng Chen. My major is computer science as everyone else in here. My expectation from this project is to have a better understanding on programming and how to program more effectively and professionally. Lastly, my task for this project is to create a table and assign everyone's tasks and due date for them.
- Matthew Kabat** 7:16 PM: Hey, my name is Mathew Kabat. I'm primarily interested in embedded systems and security, but I'm also interested more generally in most things CS. I expect to become more experienced at working with a team of fellow developers through both this and following projects.
- Aaja Christie** 10:21 PM: Hey my name is Aaja Christie. I am interested in cyber security and computer engineering. My expectation is to better understand Assembly Programming on the raspberry pi and to be able to apply this new found skill set to other projects I do in the future. My task on the assignment was creating this slack page as well as the Github and putting the Raspian OS onto the Rasberrv pi.