Developing Soft and Parallel Programming Skills Using Project-Based Learning

Spring-2019 **JupitorInk**

Nhi Ong, Alejandro Herbener, Alec Buchinski, Johnathan Moore

Table of Contents

I. Project Description

1.1 Purpose of the Project

The purpose of this project is to 1) become familiar with unified, team-based project management tools and communication applications in order to develop soft skills, 2) identify Raspberry Pi 3+ components and architecture; explain basic multicore architectures and programming using OpenMP and C language 3) gain more practice with ARM assembly programming (Mussa, Assignment 2 Worksheet).

II. Introduction to the Team

2.1 Members' Roles and Assignments

Coordinator: Johnathan Moore

For this assignment, he was the coordinator; his responsibilities contained tasks one, two and six. Task one involves group planning and scheduling and creating a table for task breakdowns. A new GitHub repository was created for assignment 3, sent out invites to the slack and access to upload to the GitHub as well. The final duties involved editing and uploading the video for part 6 to the YouTube channel.

Team Member: Alec Buchinski

This team member is responsible for task five. I created this report after the other members completed their sections and compiled them into this document.

Team member: Nhi Ong

This team member is responsible for task three. Nhi completed the Parallel Programming portion of this project, as well as taking screenshots and reporting their findings.

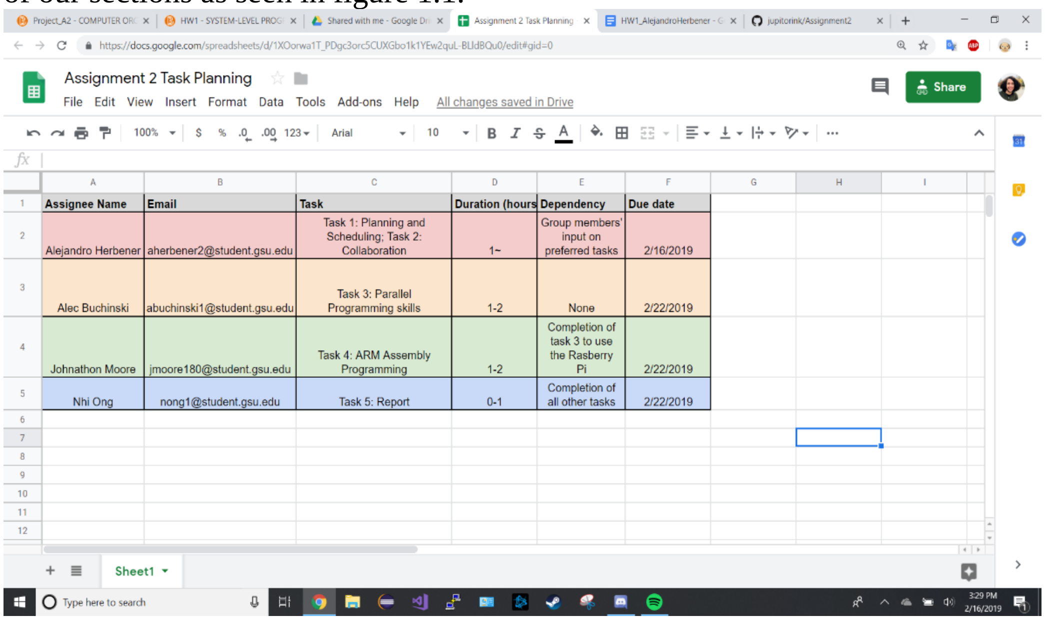
Team Member: Alejandro Herbener

This team member is responsible for task four. Alejandro completed the ARM Assembly Programming section of this project. This includes taking screenshots, completing the assigned work, and creating a write-up to send to the GitHub for further editing.

III. Planning and Scheduling

3.1 Role as Coordinator

As the coordinator, Johnathan completed the new assignment table and created a new repository for the assignment. He contacted us to discuss our preferred roles over Slack, and we selected our preferred roles and quickly started to work on each of our sections as seen in figure 1.1.



Assignee Name	Email	Task	Duration (hours	Dependency	Due date
Alejandro Herbener	aherbener2@student.gsu.edu	Task 1: Planning and Scheduling; Task 2: Collaboration	1~	Group members' input on preferred tasks	2/16/2019
Alec Buchinski	abuchinski1@student.gsu.edu	Task 3: Parallel Programming skills	1-2	None	2/22/2019
Johnathon Moore	jmoore180@student.gsu.edu	Task 4: ARM Assembly Programming	1-2	Completion of task 3 to use the Rasberry Pi	2/22/2019
Nhi Ong	nong1@student.gsu.edu	Task 5: Report	0-1	Completion of all other tasks	2/22/2019

Figure 1.1 – Task One, Assignment Chart

3.2 Additional Tasks

Alejandro was responsible for creating a new repository for this assignment with a readme page. He added a projects page which details our tasks and allows for any member to update their status on their role. (see Figure 1.2) In addition to this, he also edited our video and uploaded it to YouTube. (see Appendix A, for link to video)

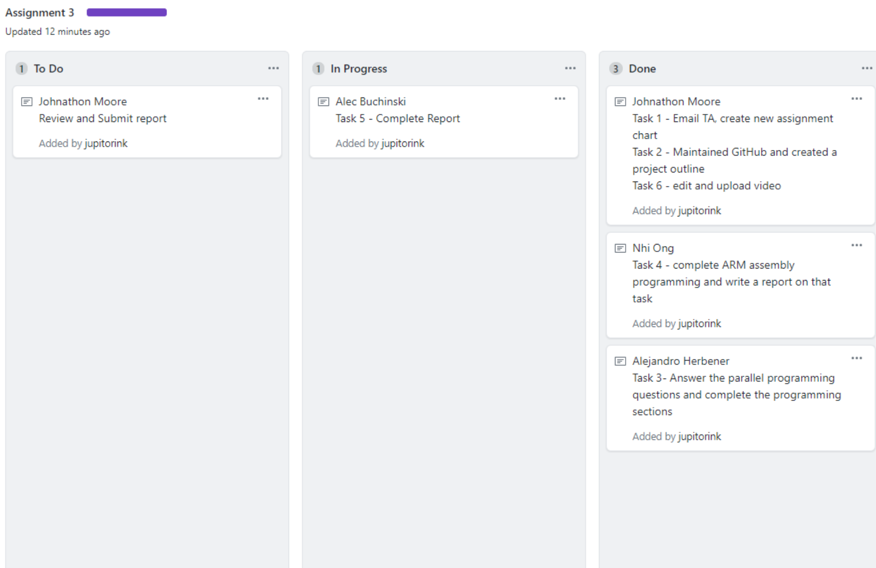


Figure 1.2 – Task Two, GitHub Projects Page Screenshot

IV. Parallel Programming Skills

Task 3a.

Define the following: Task, Pipelining, Shared Memory, Communications, Synchronization. (in your own words)

A task is a program or set of instructions that is executed by the processor to do a certain thing. Pipelining is a type of parallel computation where the work is broken down into smaller steps. Shared Memory is a type of architecture where all of the processors share access to the same physical memory. Communications is the exchange of data between different parallel processes. Synchronization deals with the coordination of tasks in parallel computation.

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

Single Instruction, Single Data is when only one instruction stream is being acted on by the CPU during any one clock cycle and only one data stream is being used as input during any one clock cycle. Single Instruction, Multiple Data has all processing units execute the same instruction at any given clock cycle and each processing unit can operate on a different data element. Multiple Instruction, Single Data is when each processing unit operates on the data independently via separate instruction streams and a single data stream is fed into multiple processing units. In Multiple Instruction, Multiple Data, every processor may be executing a different instruction stream and every processor may be working with a different data stream.

What are the Parallel Programming Models?

The Parallel Programming models are Shared memory, threads, distributed memory/message passing, data parallel, hybrid, single program multiple data, multiple program multiple data.

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

Uniform Memory Access has identical processors, equal access and access times to memory. Non-Uniform Memory Access is where one SMP can directly access of another SMP, not all processors have equal access time to all memories. OpenMP can use both types of architectures so that the code can be as portable as possible.

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

In the Shared Memory Model, processes and tasks share a common space in the memory which they use read and write at the same time, while in the Threads Model, work is split up into several smaller threads that are executed at the same time.

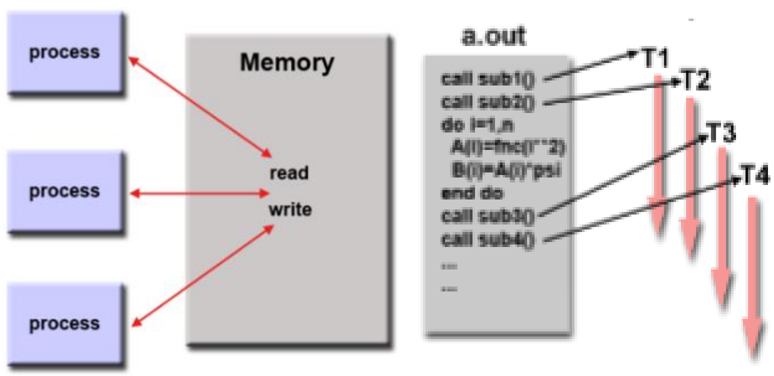


Figure 5.1– Task Three, Memory Example.

What is Parallel Programming? (in your own words)

Parallel programming is a way of programming in which multiple cores of the system are used to perform multiple tasks at the same time.

What is system on chip (SoC)? Does Raspberry PI use system on SoC?
System on chip is when the CPU, RAM, and GPU are all squeezed into one component, usually a single silicon chip. The Raspberry Pi utilizes the SoC.

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

A CPU is slightly more compact than a SoC, but the SoC contains a lot more functionality. A SoC also consumes much less power than a CPU, and also much cheaper. The biggest advantage of a CPU over a SoC is that it is much more flexible, in that new components can be put in at any time.

5.2 Task 3b.

Figure 5.2 – Task Three, Parallel Programming Screenshot A

This was the output after running the first code segment. In this iteration of the code, the number of iterations evenly divides the number of threads, and each thread performs an equal number of iterations.

```
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 2 performed iteration 11
Thread 2 performed iteration 12
Thread 2 performed iteration 13
Thread 2 performed iteration 14
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 15
```

Figure 5.3 – Task Three, Parallel Programming Screenshot B

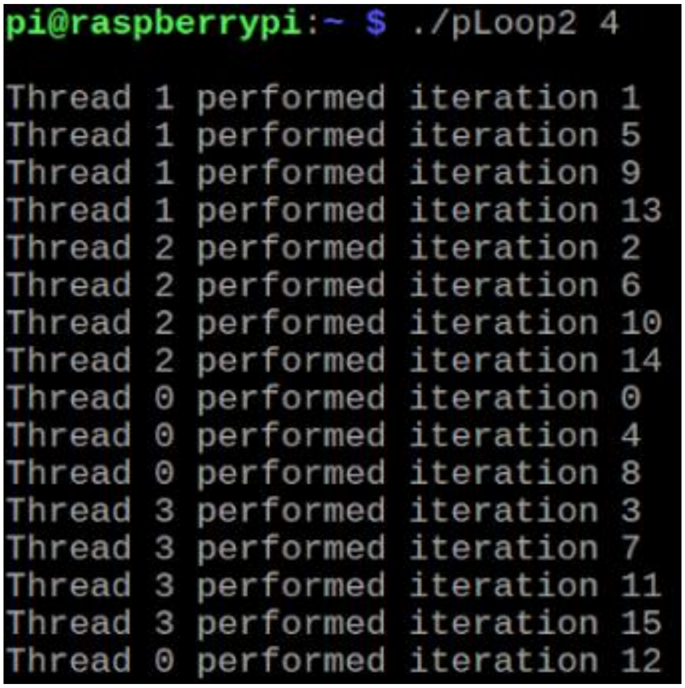
When the number of iterations is not evenly divided by the number of threads, some threads will perform less iterations than others. In this case, Thread 0 performed 6 iterations, while Threads 1 and 2 both performed 5.

```
pi@raspberrypi:~ $ ./pLoop 5

Thread 3 performed iteration 10
Thread 0 performed iteration 0
Thread 0 performed iteration 1
Thread 0 performed iteration 2
Thread 0 performed iteration 3
Thread 2 performed iteration 7
Thread 2 performed iteration 8
Thread 2 performed iteration 9
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 1 performed iteration 4
Thread 1 performed iteration 5
Thread 1 performed iteration 6
```

Figure 5.4 – Task Three, Parallel Programming Screenshot C

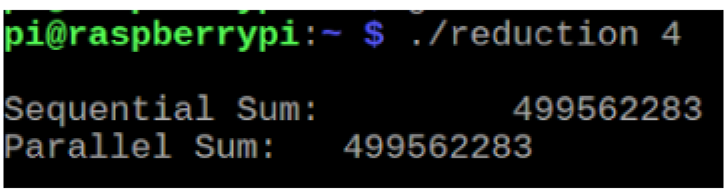
When performed with a different uneven number, Thread 0 is still the thread performing the most iterations.



```
pi@raspberrypi:~ $ ./pLoop2 4

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
Thread 0 performed iteration 0
Thread 0 performed iteration 4
Thread 0 performed iteration 8
Thread 3 performed iteration 3
Thread 3 performed iteration 7
Thread 3 performed iteration 11
Thread 3 performed iteration 15
Thread 0 performed iteration 12
```

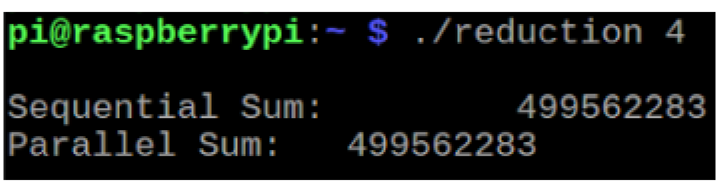
Figure 5.5 – Task Three, Parallel Programming Screenshot D



```
pi@raspberrypi:~ $ ./reduction 4

Sequential Sum:          499562283
Parallel Sum:    499562283
```

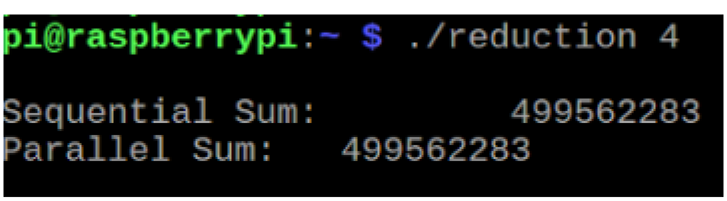
Figure 5.6 – Task Three, Parallel Programming Screenshot E



```
pi@raspberrypi:~ $ ./reduction 4

Sequential Sum:          499562283
Parallel Sum:    499562283
```

Figure 5.7 – Task Three, Parallel Programming Screenshot F



```
pi@raspberrypi:~ $ ./reduction 4

Sequential Sum:          499562283
Parallel Sum:    499562283
```

Figure 5.8 – Task Three, Parallel Programming Screenshot G

With the second code, the difference is that the assignment of iterations to threads is instead done in rounds, such as Thread 0 performing iteration 0, Thread 1 performs 1, 2 performs 2, Thread 3 performs 3, all the way back around to Thread 0 which completes iteration 4. This is done instead of each thread performing a specific range of iterations. Figure 5.5 shows the output after the sum code is run for the first time. After the commend mark is removed, figure 5.6 shows that the output remains the same as seen in figure 5.5. Figure 5.7 shows the result of the program after all comment marks were removed. The output remained the same, which was not the expected result. This was double checked several times for this reason.

V. ARM Assembly Programming

03/04 – Raspberry Pi was received from teammate
03/05 – Work on the ARM Assembly portion of the assignment was started. Nhi had to familiarize herself with the Raspberry Pi at first, by figuring out how to use gdb, terminal commands, and Scrot to take screenshots of the Pi.
03/06 – Nhi formally began work on the program.

Part 1.

What error messages did you get and why?

The screen reported the following error message, as seen in figure 6.1: 'Error: unknown pseudo-op: '.shalfword'
This error was due to the use of an incorrect data representation in the program. This was corrected by changing '.shalfword' to '.hword'

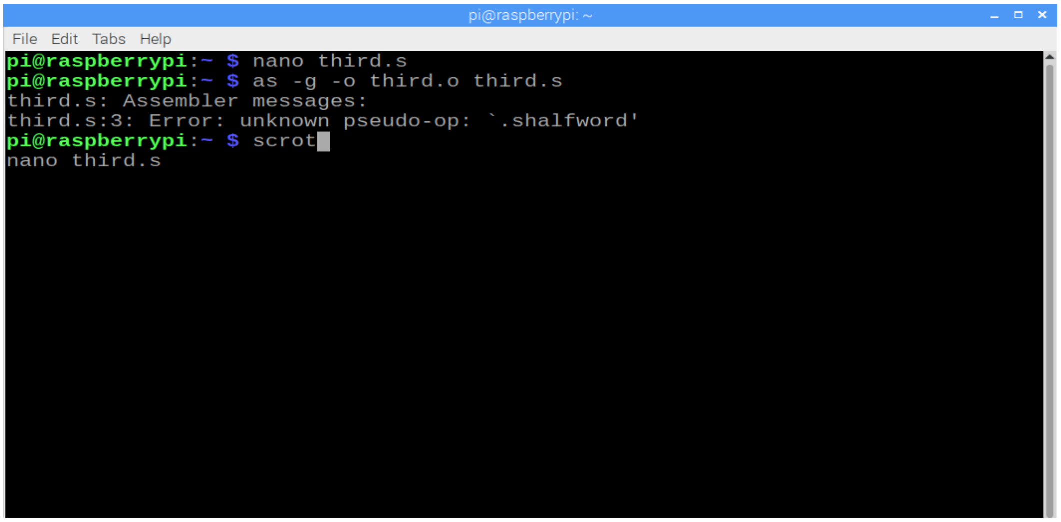


Figure 6.1 – Task Four, ARM Assembly Programming Screenshot A

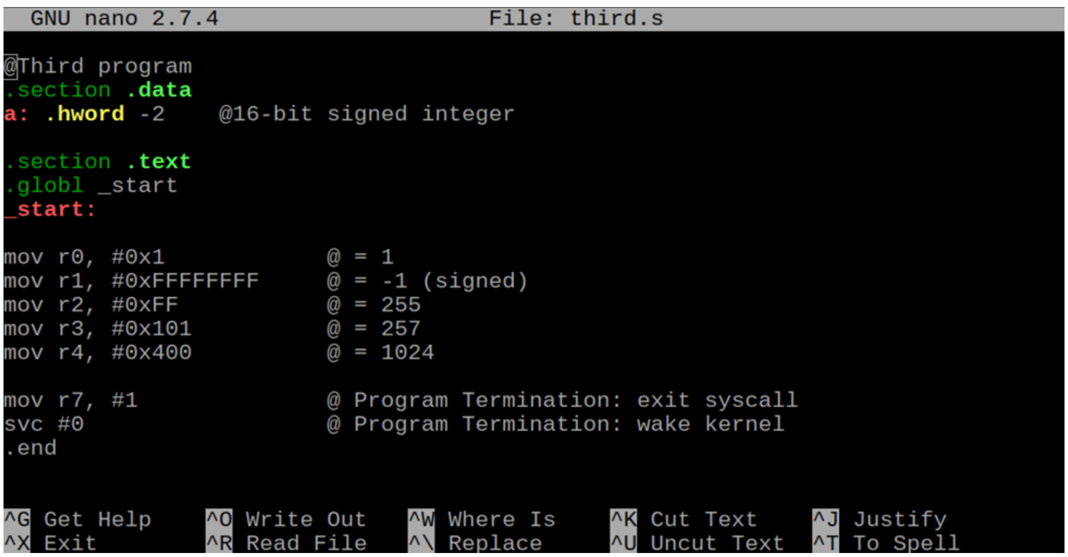


Figure 6.2 – Task Four, ARM Assembly Programming Screenshot B

Nhi debugged the program using GDB. The program is visible above in figure 6.2. She first used list to display the code then set a breakpoint and stepped through the program. She used ‘p &a’ to retrieve the address of the halfword integer. As instructed in class, she used the ‘x’ command to examine the memory. She did this for both ‘h’ and ‘sh’ as seen in figure 6.3.

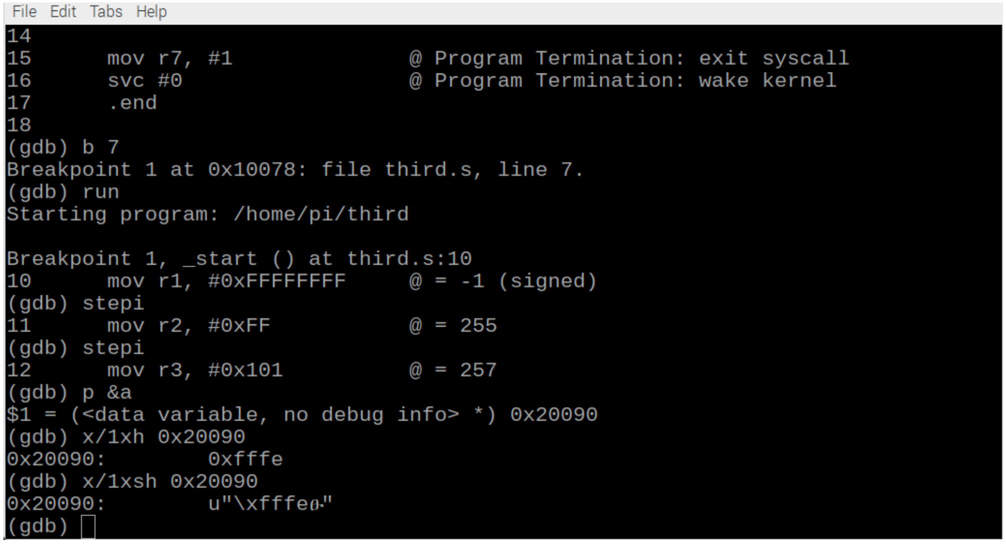


Figure 6.3 – Task Four, ARM Assembly Programming Screenshot C

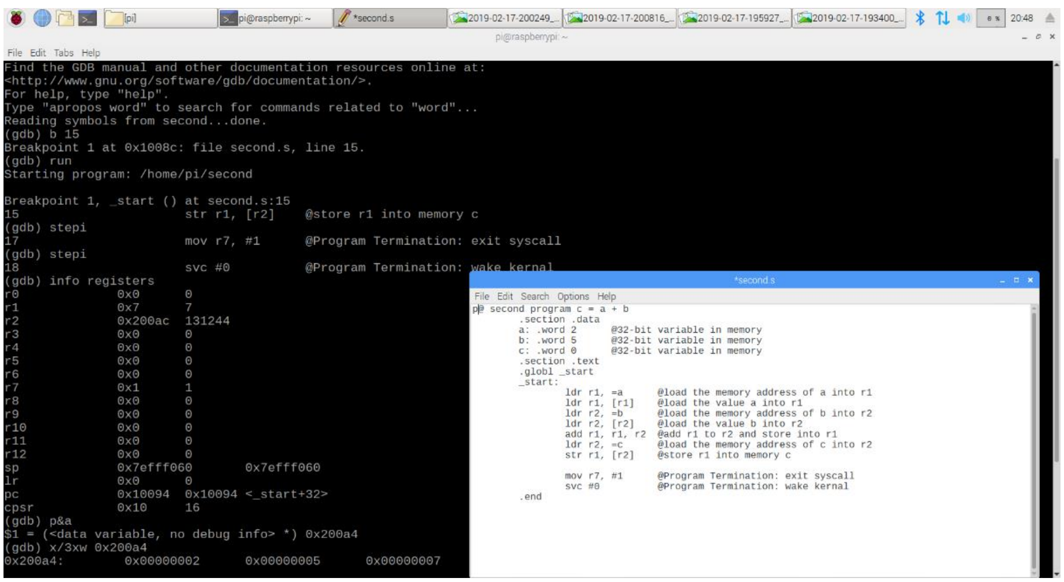


Figure 6.4 – Task Four, Raspberry Pi Screenshot E

Part 2.

```
File  Edit  Tabs  Help
  GNU nano 2.7.4                          File: third.s

@Arithmetic program
.section .data
val1: .byte -60         @8-bit unsigned integer
val2: .byte 11          @8-bit unsigned integer
val3: .byte 16          @8-bit signed integer

.section .text
.globl _start
_start:

ldrb r0, =val1
ldrb r0, [r0]
ldrb r2, =val2
ldrb r2, [r2]
ldrsb r3, =val3
ldrsb r3, [r3]

add r2, r2, #3
add r2, r2, r3
sub r2, r2, r0

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

                              [ Read 25 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
```

Figure 6.5 – Task Four, Raspberry Pi Screenshot D

For part two, Nhi created a constant for val1, val2, and val3 in the .data portion of the program. In order to use the constant values, she used ‘ldr’ to load the values into memory. For unsigned bytes, she used ‘ldrb’ and ‘ldrsb’ for signed bytes. She then completed the arithmetic and worked through the program in GDB. She then stepped through the program and checked registers for the ending values. As shown in figure 6.6, ‘cpsr’ shows that the flag was indeed raised.

```
(gdb) b 7
Breakpoint 1 at 0x10078: file third.s, line 7.
(gdb) run
Starting program: /home/pi/third

Breakpoint 1, _start () at third.s:10
10
(gdb) stepi
11      ldrb r0, =val1
(gdb) step i
No symbol "i" in current context.
(gdb) stepi
12      ldrb r0, [r0]
(gdb) info registers
r0             0x1      1
r1             0xffffffff       4294967295
r2             0xff     255
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             0x10080  0x10080 <_start+12>
cpsr           0x10     16
(gdb) 
```

Figure 6.6 – Task Four, Raspberry Pi Screenshot F

VI. Appendix

A. Link to Slack, GitHub, YouTube Channel, YouTube Video

Slack:
https://csc-3210.slack.com

GitHub:
https://github.com/jupitorink

GitHub Assignment One Repository:
https://github.com/jupitorink/Assignment-3

YouTube Channel:
https://www.youtube.com/channel/UChTgBqHzG9G-Mvv-q7uK96g

YouTube Video:
https://www.youtube.com/watch?v=eWah_4RSt8M&feature=youtu.be

B. Task One Screenshot

Task Assignment Table:
https://github.com/jupitorink/Assignment-3/blob/master/Task%20Table.png

C. Task Two Screenshot

GitHub Projects Page:
https://github.com/jupitorink/Assignment-3/projects/1

D. Task Three Parallel Programming Report, Screenshots

Parallel Programming Report
https://github.com/jupitorink/Assignment3/blob/master/Assignment_3_Task_3.docx

E. Task Four ARM Assembly Programming Report, Screenshots

ARM Assembly Programming Report
https://github.com/jupitorink/Assignment-3/blob/master/armreport.docx

Raspberry Pi Screenshot A
https://github.com/jupitorink/Assignment-3/blob/master/armA.png

Raspberry Pi Screenshot B
https://github.com/jupitorink/Assignment-3/blob/master/armB.png

Raspberry Pi Screenshot C
https://github.com/jupitorink/Assignment-3/blob/master/armC.png

Raspberry Pi Screenshot D
https://github.com/jupitorink/Assignment-3/blob/master/armD.png

Raspberry Pi Screenshot E
https://github.com/jupitorink/Assignment-3/blob/master/armE.png