

Developing Soft and Parallel Programming Skills Using Project-Based Learning

Spring 2019

Group Name: ATLAS-SQUAD

Team Members: Jason Poston (Team Coordinator)
Sai Rampally
T'Avvion Jones
Zeak Sims
Shili Guan

Planning and Scheduling

Assignee Name	Email	Task	Duration (hours)	Dependency	Due Date	Note
Jason Poston (Team Lead)	jposton1@student.gsu.edu	Task 3 Foundation Task 3 Parallel Programing Basics Task 4 ARM Assembly Programing Task 5: Report/GitHub	4-6 hrs	None	02/19	All tasks must be finished and sent to lead by 2/19/19. Excellent (100%)
T'Avvion Jones	Tjones172@student.gsu.edu	Task 3a Q:3-5 Task 6 video Task 5: Submitting report via Google Docs/ GitHub	2 hrs	None	02/19	All tasks must be finished and sent to lead by 2/19/19. Excellent (100%)
Shili Guan	sguan2@student.gsu.edu	Task 3a Q:6,7 Task 6 video Task 5: Submitting report via Google Docs/ GitHub	2 hrs	None	02/19	All tasks must be finished and sent to lead by 2/19/19. Excellent (100%)
Zeak Sims	zsims2@student.gsu.edu	Task 3a Q:8,9 Task 5: Submitting report via Google Docs/ GitHub Task 6 video	2 hrs	None	02/19	All tasks must be finished and sent to lead by 2/19/19. Excellent (100%)
Sai Rampally	srampally1@student.gsu.edu	Task 3a Q 1+2 Task 6 (Upload YouTube Video) Assist with Google Docs/GitHub	2-3 hrs	YouTube channel Upload	02/19	All tasks must be finished and sent to lead by 2/19/19. Excellent (100%)

TeamWork Basics

1.) Identifying the components on the Raspberry PI B+

The Raspberry Pi B+ consists of a 1GB RAM, SD card slot, RCA jack, LED lights, CPU, 4 USB 2.0 ports, HDMI connection port, a 3.5 mm audio jack, Power cable slot, Camera jack, and a ethernet port/wireless internet LAN. The CPU is a 64-bit ARM multi-core, and the video output can also be connected to raspberry pi via HDMI. It supports a wireless LAN, and there is an ethernet jack which makes the pi to be able to connect through LAN wire.

2.) How many cores does the Raspberry Pi B+ CPU have

The raspberry pi has quad core processor (4 cores) that it can process multiple tasks.

3.) List three main differences between X86 (CISC) and ARM Raspberry PI (RISC).

Justify your answer and use you own words (do not copy and paste).

The main and most significant difference between the two is the instruction set. The ARM has a smaller instruction set, it also contains more registers than the X86. When compared to the X86, ARM has more operations, and more addressing modes. ARM processor has a simple instruction set, and uses more registers when compared to the X86 CISC architecture. Because of its limited instruction set, it has to do many more operations than X86. This might seem like a disadvantage, however having a reduced instruction set, allows the ARM to execute more quickly, though harder (longer) to write. In ARM, instruction can be used for conditional execution.

4.) What is the difference between sequential and parallel computation and identify the practical significance of each?

The difference between sequential and parallel computation is that in serial computing a problem is broken up into a series of instructions and executed one after the other in a sequential order on a single processor. Whereas in parallel programming, a problem is broken into small pieces that are then broken into smaller instructions that will be solved simultaneously yet executed on different processors.

5.) Identify the basic form of data and task parallelism in computational problems.

The basic form of data and task parallelism in computation problems is:

Data Parallelism is the simultaneous execution on multiple cores of the same functions across the elements of the dataset. While in task parallelism, it also simultaneous executed on multiple cores of many functions across the same of different datasets.

6.) Explain the difference between processes and threads.

A process is the abstraction of a running program that don't share memory with each other; Thread is a lightweight process that allows a single executable/process to be decomposed to smaller and independent parts. Thread shares the common memory of the process they belong to. Therefore, the difference between processes and threads is that while processes don't share memory, the threads share common memory of the process which they belong to.

7.) What is OpenMP and what is OpenMP pragmas?

OpenMP (Open Multi-Processing) is an application programming interface, it uses an implicit multithreading model in which the library handles thread creation and management. OpenMP can reduce the complex and errors of the task. OpenMP pragmas is compiler directives that enable the compiler to generate threaded code. OpenMP programs also use multiple instruction, multiple data pattern additional to the Thread pool pattern.

8.) What applications benefit from multi-core (list four)

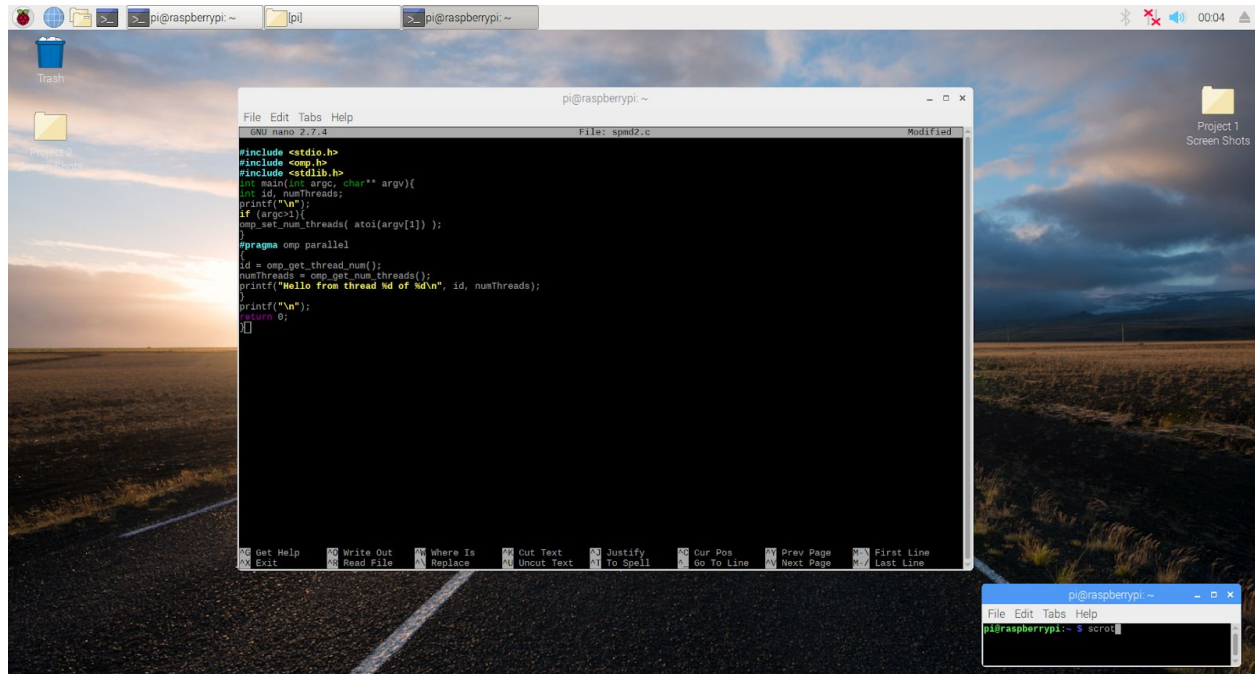
The applications that benefit from multi-core are Database servers, compilers, multimedia applications, and web services/commerce.

9.) Why Multicore?

Multi-core architectures have many advantages over the single core architecture. Most major OS support multi-core and perceive each core as a separate processor. This means the OS can execute more processes at one time also known as concurrency. There are a couple of disadvantages with the single-core architecture. For one, it is difficult to make single-core clock frequencies even higher. Many new applications also are multithread, and there has be more of a shift towards more parallelism. Another disadvantage of the single-core architecture is that it leads to overheating.

Parallel Programming Basics

The initial tasks were to become more familiar with the Raspberry Pi system, and the Terminal that we will use to code stuff in. After following along with a few instructions on how to maneuver better inside the Terminal, I began coding using the spmd2 template.

A screenshot of a Raspberry Pi desktop environment. The background is a scenic image of a field at sunset. In the center, a terminal window titled 'pi@raspberrypi: ~' is open, displaying C code in a nano editor. The code is for a parallel program using OpenMP. It includes headers for stdio, omp, and stdlib. The main function takes an argument for the number of threads. It uses omp_set_num_threads to set the number of threads and omp_get_thread_num to get the thread ID. A #pragma omp parallel block contains the main logic, which prints a message for each thread. The code is as follows:

```
#include <stdio.h>
#include <omp.h>
#include <stdlib.h>
int main(int argc, char** argv){
    int id, numThreads;
    printf("\n");
    if (argc>2){
        omp_set_num_threads( atoi(argv[1]) );
    }
    #pragma omp parallel
    {
        id = omp_get_thread_num();
        numThreads = omp_get_num_threads();
        printf("Hello from thread %d of %d\n", id, numThreads);
    }
    printf("\n");
    return 0;
}
```

At the bottom right, there is a small terminal window titled 'pi@raspberrypi: ~' showing the command 'pi@raspberrypi:~\$ screen'.

Line 10 (#pragma omp parallel) here was the focus, as it is what initiate the ‘fork’ thread. This allows line 11-15 to perform different tasks concurrently. Anything inside the “{}” would be run

The screenshot displays a Raspberry Pi desktop environment. The background is a scenic image of a field. The desktop includes a taskbar at the bottom with icons for the Pi logo, file manager, terminal, and other applications. A file manager window is open, showing the contents of the home directory. A terminal window is also open, displaying the execution of a program with multiple threads. The terminal output shows 'Hello from thread' messages for 4 threads, 2 threads, and 6 threads.

```

pi@raspberrypi:~$ scrot
pi@raspberrypi:~$ ls Downloads/
ARM_Assembly_Programming.pdf
pi@raspberrypi:~$ nano spmd2.c
pi@raspberrypi:~$ ./spmd2
bash: ./spmd2: No such file or directory
pi@raspberrypi:~$ nano spmd2.c
pi@raspberrypi:~$ gcc spmd2.c -o spmd2 -fopenmp
pi@raspberrypi:~$ ./spmd2

Hello from thread 2 of 4
Hello from thread 2 of 4
Hello from thread 2 of 4
Hello from thread 3 of 4

pi@raspberrypi:~$ ./spmd2 4

Hello from thread 2 of 4
Hello from thread 3 of 4
Hello from thread 2 of 4
Hello from thread 2 of 4
Hello from thread 2 of 4

pi@raspberrypi:~$ ./spmd2 2

Hello from thread 0 of 2
Hello from thread 1 of 2

pi@raspberrypi:~$ ./spmd2 6

Hello from thread 1 of 6
Hello from thread 3 of 6
Hello from thread 2 of 6
Hello from thread 4 of 6
Hello from thread 0 of 6
Hello from thread 5 of 6

pi@raspberrypi:~$

```


was being handled by one and not multiple cores.

The image consists of two screenshots of a Raspberry Pi desktop environment. The desktop background is a landscape photo of a sunset over a field. The top panel shows the system menu with icons for Trash, Home, and Pi. The bottom panel shows the system status with the time 00:18 and battery level. The main window is a file manager showing the contents of the /home/pi directory, including folders like Downloads, Music, and Pictures, and files like first.s, spmd2, and spmd2.c. A terminal window is open in the foreground, displaying the output of a program. The program is a parallel execution of a simple 'Hello from thread' message. The first screenshot shows the program running with 6 threads, and the second screenshot shows it running with 4 threads. The output is interleaved, indicating that the threads are not running sequentially. The terminal also shows the compilation of the program using gcc and the execution of the resulting binary.

```
#include <stdio.h>
#include <omp.h>
#include <stdlib.h>
int main(int argc, char** argv){
    // int id, numThreads;
    printf("\n");
    if (argc>2){
        omp_set_num_threads( atoi(argv[1]) );
    }
    #pragma omp parallel
    {
        int id = omp_get_thread_num();
        int numThreads = omp_get_num_threads();
        printf("Hello from thread %d of %d\n", id, numThreads);
    }
    printf("\n");
    return 0;
} // Revised
```

```
pi@raspberrypi:~$ ./spmd2 6
Hello from thread 0 of 6
Hello from thread 1 of 6
Hello from thread 2 of 6
Hello from thread 3 of 6
Hello from thread 4 of 6
Hello from thread 5 of 6

pi@raspberrypi:~$ ./spmd2 4
Hello from thread 1 of 4
Hello from thread 0 of 4
Hello from thread 3 of 4
Hello from thread 2 of 4

pi@raspberrypi:~$ ./spmd2 4
Hello from thread 2 of 4
Hello from thread 1 of 4
Hello from thread 3 of 4
Hello from thread 0 of 4

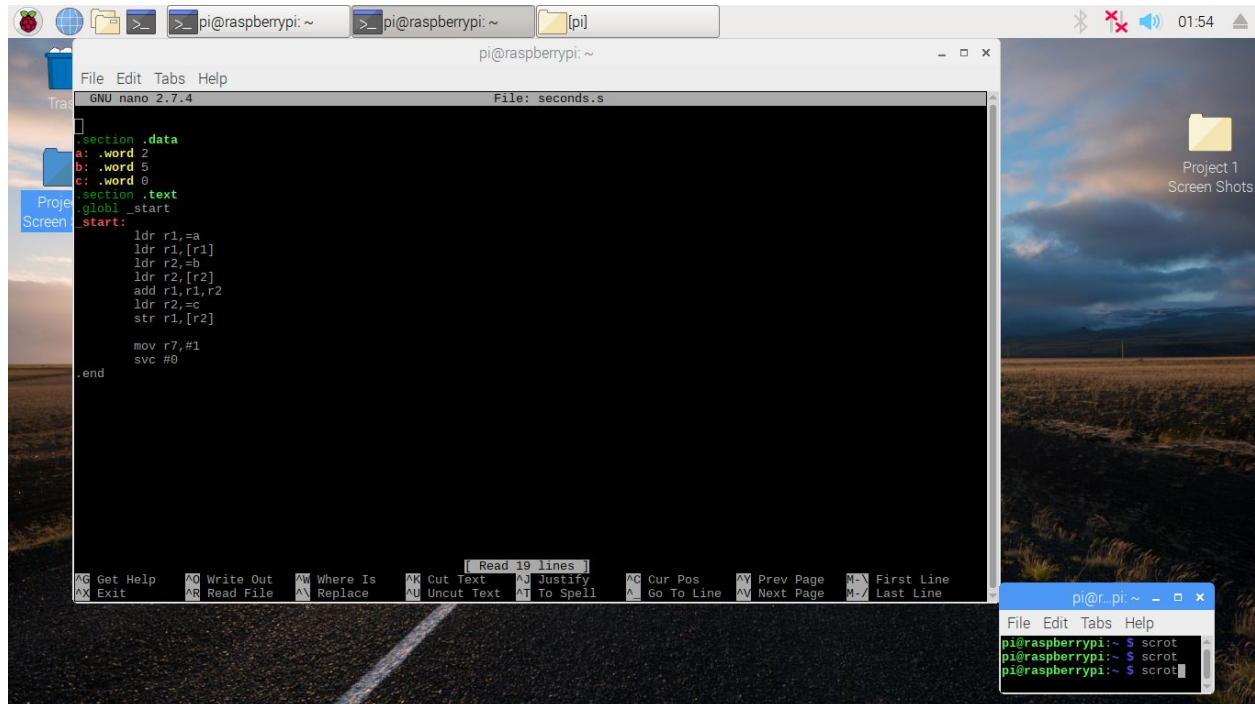
pi@raspberrypi:~$ nano spmd2.c
pi@raspberrypi:~$ gcc spmd2.c -o spmd2 -fopenmp
pi@raspberrypi:~$ ./spmd2 4
Hello from thread 0 of 4
Hello from thread 1 of 4
Hello from thread 2 of 4
Hello from thread 3 of 4

pi@raspberrypi:~$
```

Now, while they're not in sequential order, only one core was used per thread, no matter how many I typed in (tried a few just to see).

ARM Assembly Programming

The second task was to copy the code from the handout and link it.



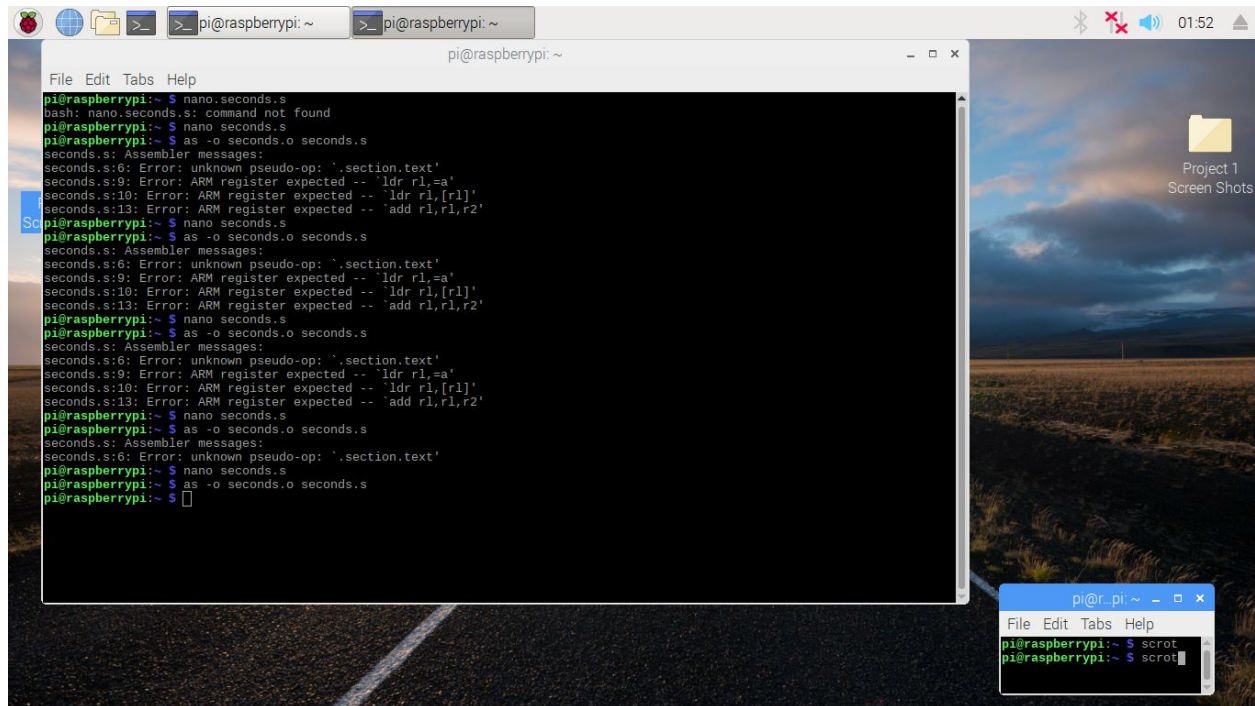
The screenshot displays a Raspberry Pi desktop with a terminal window open, running the GNU nano 2.7.4 text editor. The editor is editing a file named 'seconds.s'. The code in the file is as follows:

```
.section .data
a: .word 2
b: .word 5
c: .word 0
.section .text
globl _start
_start:
    ldr r1,=a
    ldr r1,[r1]
    ldr r2,=b
    ldr r2,[r2]
    add r1,r1,r2
    ldr r2,=c
    str r1,[r2]

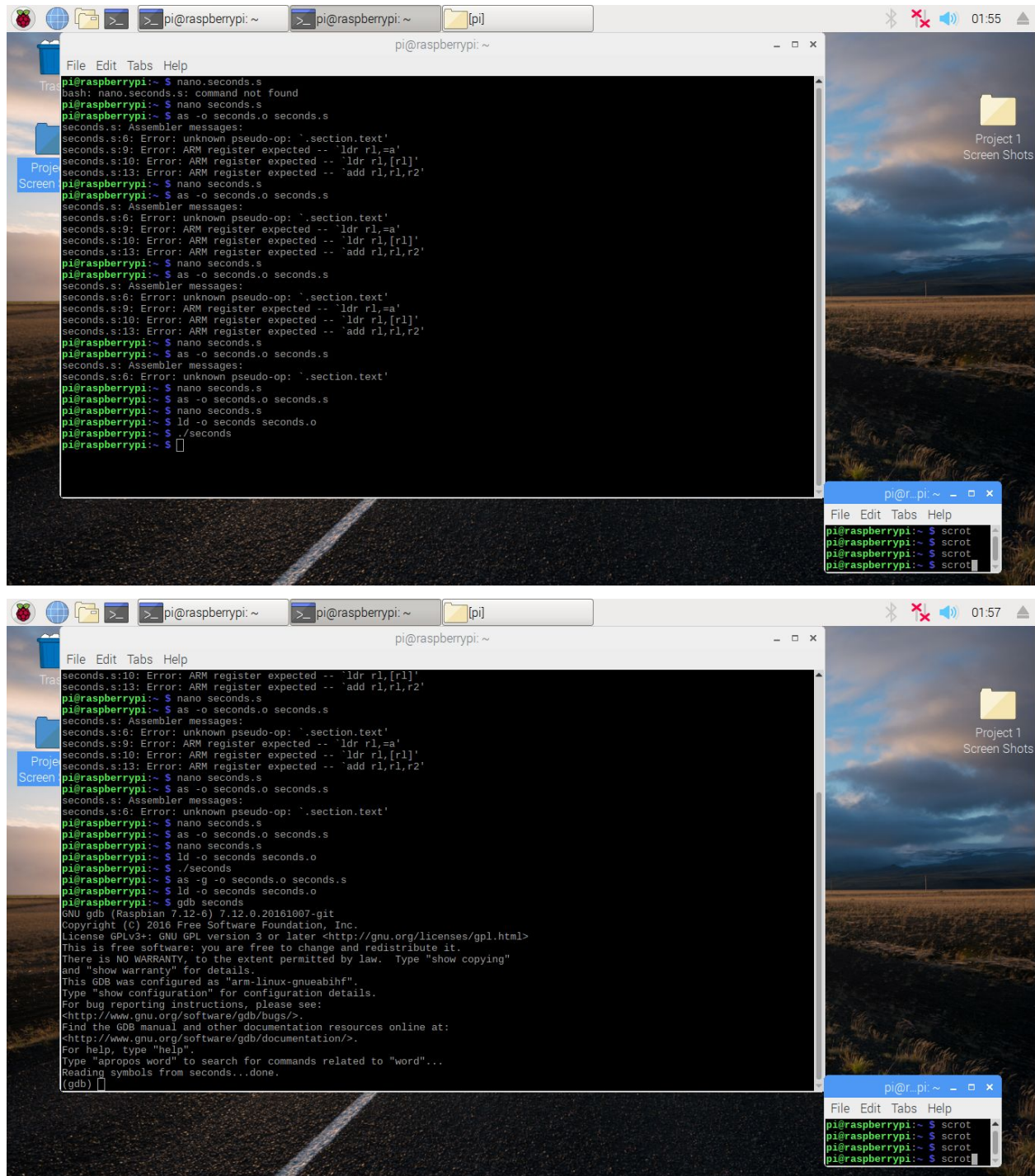
    mov r7,#1
    svc #0
.end
```

The terminal window in the bottom right corner shows the command 'scrot' being entered, which is used to take a screenshot of the desktop.

Once linked, the program ran, however didn't produce any output. We needed run the debugger to see what was going on with the actual program.

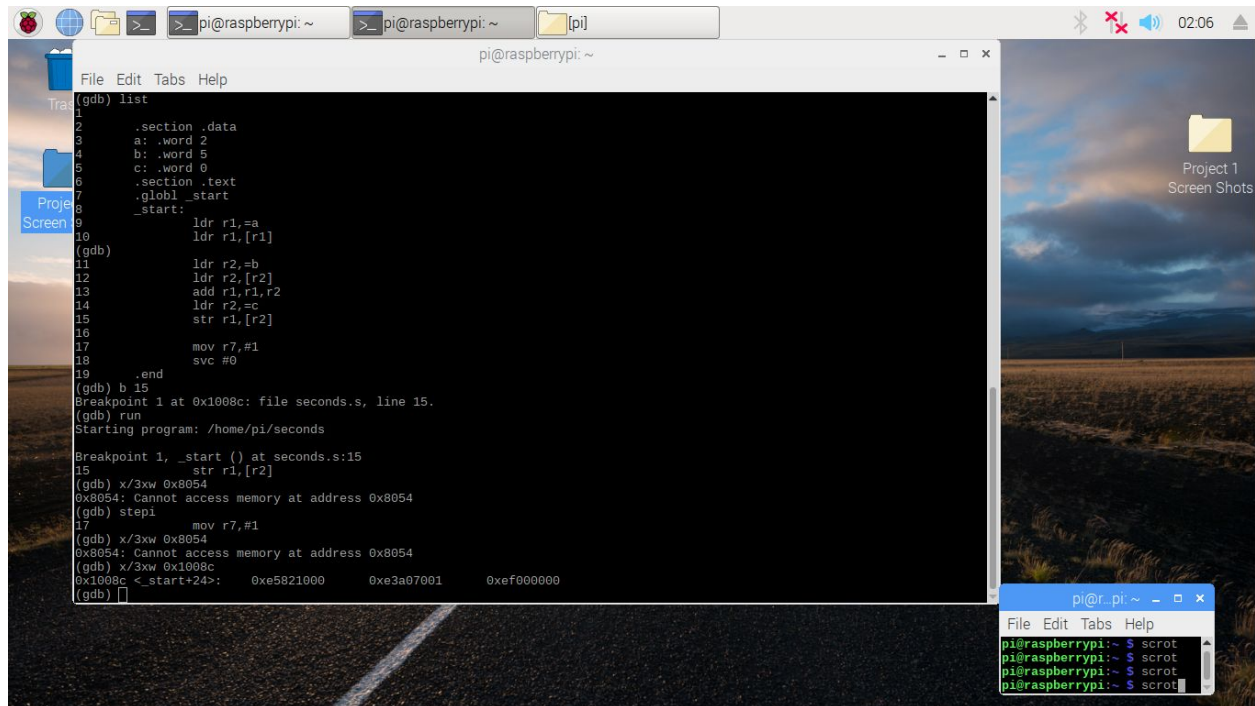


After adding the flags to the assembler line command, linking it again, and running the debugger we were able to apply breakpoints to have a better understanding of what was being stored and where.



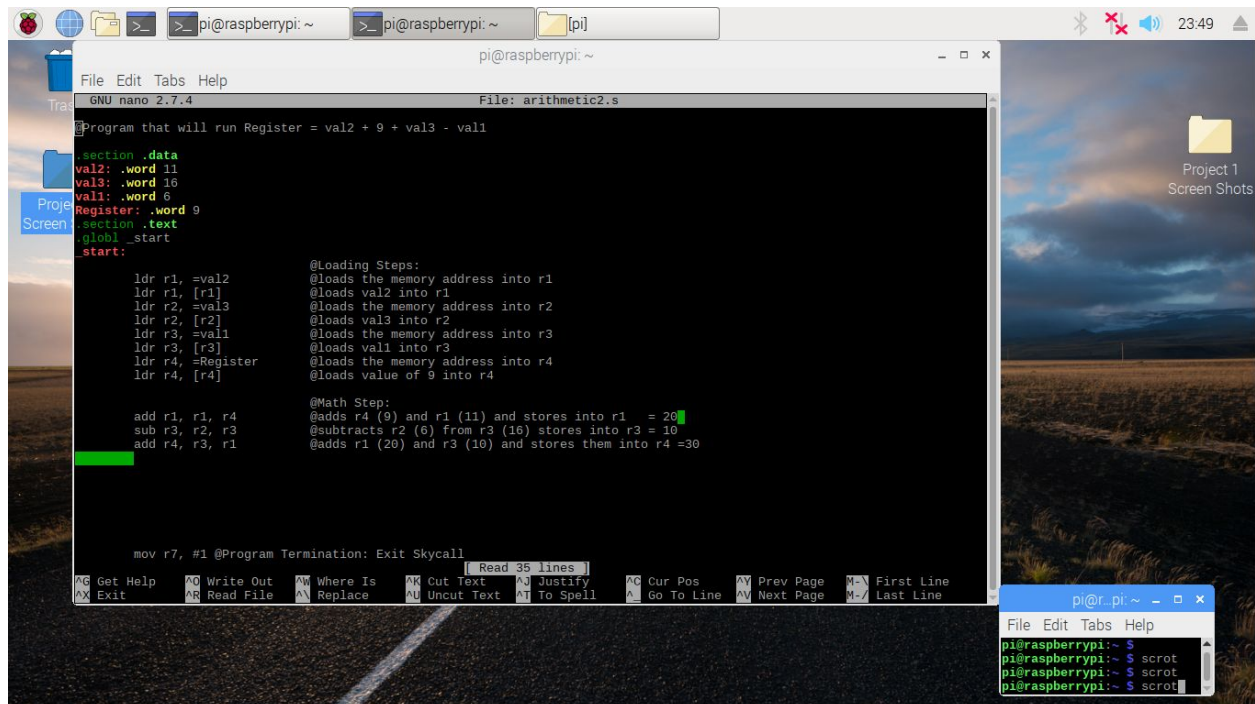
The breakpoint was applied to Line 15 (the Storing step). Here we learned how to read and ask for certain information from the register. You can ask for the format and size in various way (like hexbyte or bitword). In this exercise we were curious about the Hex of our register which was 0x1008c, displayed below. This allowed us to see what was stored in there after the breakpoint which allows us to better follow and see where the values are in our memory.

0x35821000



```
(gdb) list
1
2  .section .data
3  a: .word 2
4  b: .word 5
5  c: .word 0
6
7  .section .text
8  .globl _start
9  _start:
10     ldr r1,=a
11     ldr r1,[r1]
12
13     ldr r2,=b
14     ldr r2,[r2]
15     add r1,r1,r2
16     ldr r2,=c
17     str r1,[r2]
18     mov r7,#1
19     svc #0
20 .end
(gdb) b 15
Breakpoint 1 at 0x1008c: file seconds.s, line 15.
(gdb) run
Starting program: /home/pi/seconds
Breakpoint 1, _start () at seconds.s:15
15     str r1,[r2]
(gdb) x/3xw 0x8054
0x8054: Cannot access memory at address 0x8054
(gdb) stepi
17     mov r7,#1
(gdb) x/3xw 0x8054
0x8054: Cannot access memory at address 0x8054
(gdb) x/3xw 0x1008c
0x1008c <_start+24>:  0xe5821000    0xe3a07001    0xef000000
(gdb)
```

The second part of the assignment was to write our own code that would add values together and store them in a register.



```
GNU nano 2.7.4 File: arithmetic2.s
Program that will run Register = val2 + 9 + val3 - val1

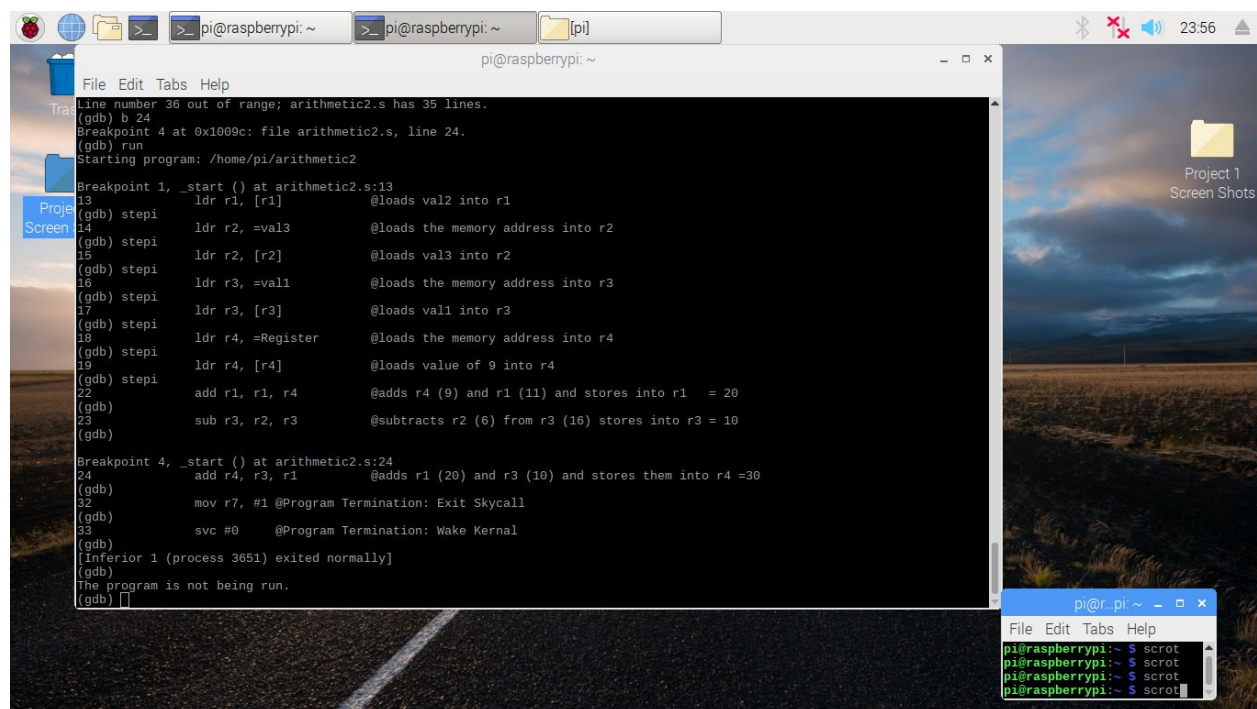
.section .data
val2: .word 11
val3: .word 16
val1: .word 6
Register: .word 9
.section .text
.globl _start
_start:

    @Loading Steps:
    ldr r1,=val2      @loads the memory address into r1
    ldr r1,[r1]       @loads val2 into r1
    ldr r2,=val3      @loads the memory address into r2
    ldr r2,[r2]       @loads val3 into r2
    ldr r3,=val1      @loads the memory address into r3
    ldr r3,[r3]       @loads val1 into r3
    ldr r4,=Register  @loads the memory address into r4
    ldr r4,[r4]       @loads value of 9 into r4

    @Math Step:
    add r1, r1, r4     @adds r4 (9) and r1 (11) and stores into r1 = 20
    sub r3, r2, r3     @subtracts r2 (6) from r3 (16) stores into r3 = 10
    add r4, r3, r1     @adds r1 (20) and r3 (10) and stores them into r4 =30

    mov r7, #1 @Program Termination: Exit Skycall
```


Again, we were to write, assemble, link and debug the code



The screenshot shows a Raspberry Pi desktop with a dark background. A large window titled 'pi@raspberrypi: ~' displays the GDB debugger interface. The window has a menu bar with 'File', 'Edit', 'Tabs', and 'Help'. The main area shows the following text:

```
Line number 36 out of range; arithmetic2.s has 35 lines.
(gdb) b 24
Breakpoint 4 at 0x1009c: file arithmetic2.s, line 24.
(gdb) run
Starting program: /home/pi/arithmetic2
Breakpoint 1, _start () at arithmetic2.s:13
13      ldr r1, [r1]          @loads val2 into r1
(gdb) stepi
14      ldr r2, =val3        @loads the memory address into r2
(gdb) stepi
15      ldr r2, [r2]          @loads val3 into r2
(gdb) stepi
16      ldr r3, =val1        @loads the memory address into r3
(gdb) stepi
17      ldr r3, [r3]          @loads val1 into r3
(gdb) stepi
18      ldr r4, =Register    @loads the memory address into r4
(gdb) stepi
19      ldr r4, [r4]          @loads value of 9 into r4
(gdb) stepi
22      add r4, r1, r4        @adds r4 (9) and r1 (11) and stores into r1 = 20
(gdb)
23      sub r3, r2, r3        @subtracts r2 (6) from r3 (16) stores into r3 = 10
(gdb)
Breakpoint 4, _start () at arithmetic2.s:24
24      add r4, r3, r1        @adds r1 (20) and r3 (10) and stores them into r4 =30
(gdb)
32      mov r7, #1 @Program Termination: Exit Skycall
(gdb)
33      svc #0 @Program Termination: Wake Kernal
(gdb)
[Inferior 1 (process 3651) exited normally]
(gdb)
The program is not being run.
(gdb)
```

On the right side of the desktop, there is a folder icon labeled 'Project 1 Screen Shots'. In the bottom right corner, there is a smaller terminal window titled 'pi@r. pi: ~' with a menu bar 'File Edit Tabs Help' and the following text:

```
pi@raspberrypi:~$ scrot
pi@raspberrypi:~$ scrot
pi@raspberrypi:~$ scrot
pi@raspberrypi:~$ scrot
```

Then calling back what we just learned, obtain the Hex code from the memory address of 0x1009c, which was 0xe0834001. Again, we were able to observe the value located at the specific address using the debugger. Looking back at the pervious project, we also checked using info register prompt to bring up a list and their stored values. This ensured that what expected to be there was in fact there.

Appendix

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

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

YouTube link: <https://www.youtube.com/watch?v=L50Ig93PXsY>