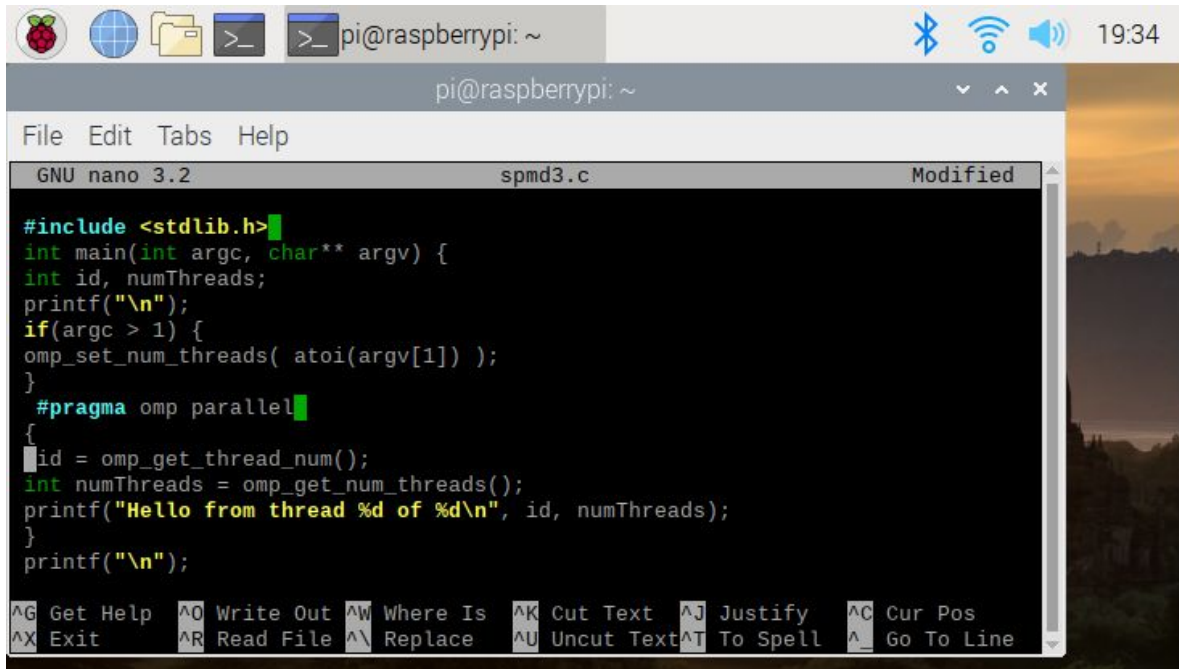
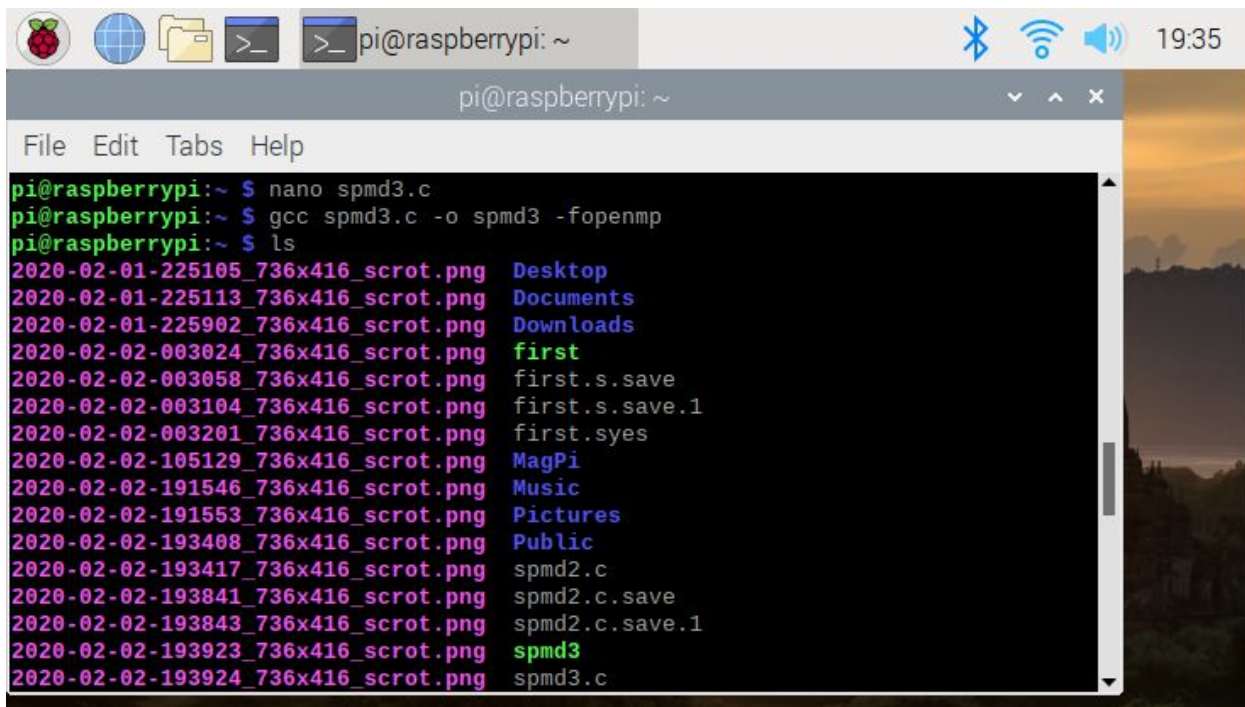


PARALLEL PROGRAMMING



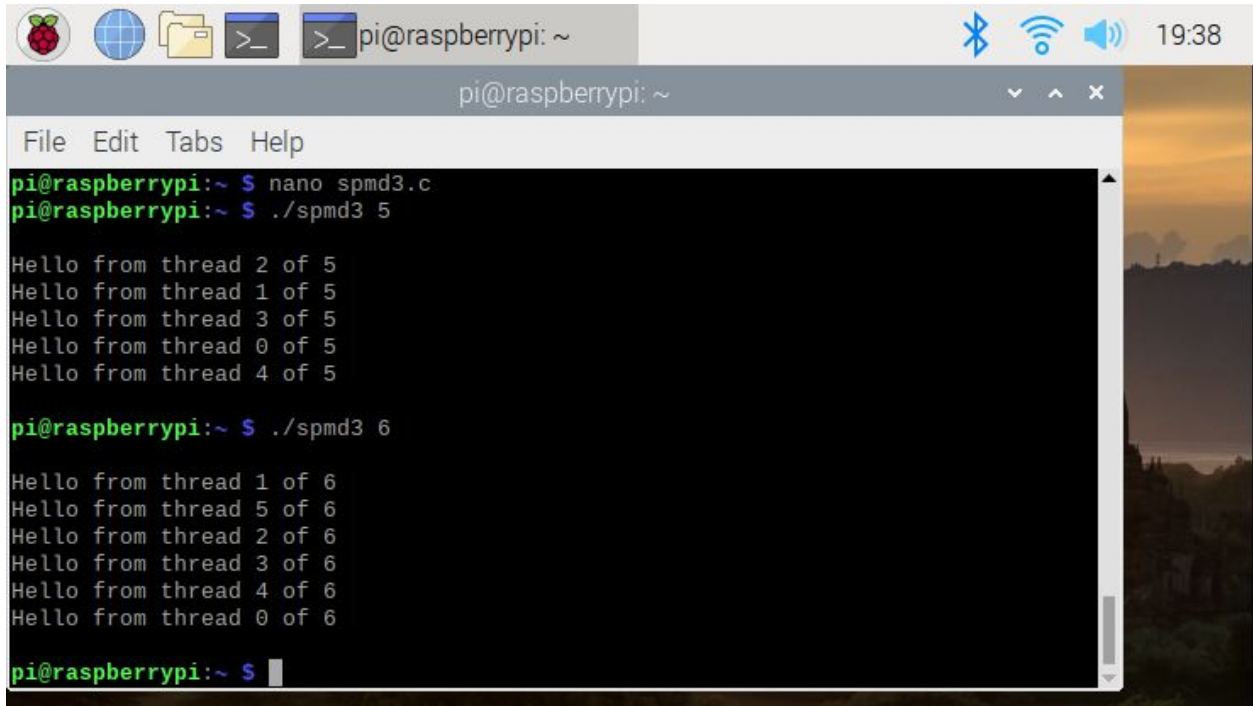
```
pi@raspberrypi: ~  
File Edit Tabs Help  
GNU nano 3.2 spmd3.c Modified  
  
#include <stdlib.h>  
int main(int argc, char** argv) {  
    int id, numThreads;  
    printf("\n");  
    if(argc > 1) {  
        omp_set_num_threads( atoi(argv[1]) );  
    }  
    #pragma omp parallel  
    {  
        id = omp_get_thread_num();  
        int numThreads = omp_get_num_threads();  
        printf("Hello from thread %d of %d\n", id, numThreads);  
    }  
    printf("\n");  
  
^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 the original source code for spmd3.c. I saved my file differently compared to the guidelines, that is why I have spmd3 instead of spmd2.c.



```
pi@raspberrypi: ~  
File Edit Tabs Help  
  
pi@raspberrypi:~ $ nano spmd3.c  
pi@raspberrypi:~ $ gcc spmd3.c -o spmd3 -fopenmp  
pi@raspberrypi:~ $ ls  
2020-02-01-225105_736x416_scrot.png Desktop  
2020-02-01-225113_736x416_scrot.png Documents  
2020-02-01-225902_736x416_scrot.png Downloads  
2020-02-02-003024_736x416_scrot.png first  
2020-02-02-003058_736x416_scrot.png first.s.save  
2020-02-02-003104_736x416_scrot.png first.s.save.1  
2020-02-02-003201_736x416_scrot.png first.syes  
2020-02-02-105129_736x416_scrot.png MagPi  
2020-02-02-191546_736x416_scrot.png Music  
2020-02-02-191553_736x416_scrot.png Pictures  
2020-02-02-193408_736x416_scrot.png Public  
2020-02-02-193417_736x416_scrot.png spmd2.c  
2020-02-02-193841_736x416_scrot.png spmd2.c.save  
2020-02-02-193843_736x416_scrot.png spmd2.c.save.1  
2020-02-02-193923_736x416_scrot.png spmd3  
2020-02-02-193924_736x416_scrot.png spmd3.c
```

In this screenshot, I used ls to check the current directories and make sure it is saved.



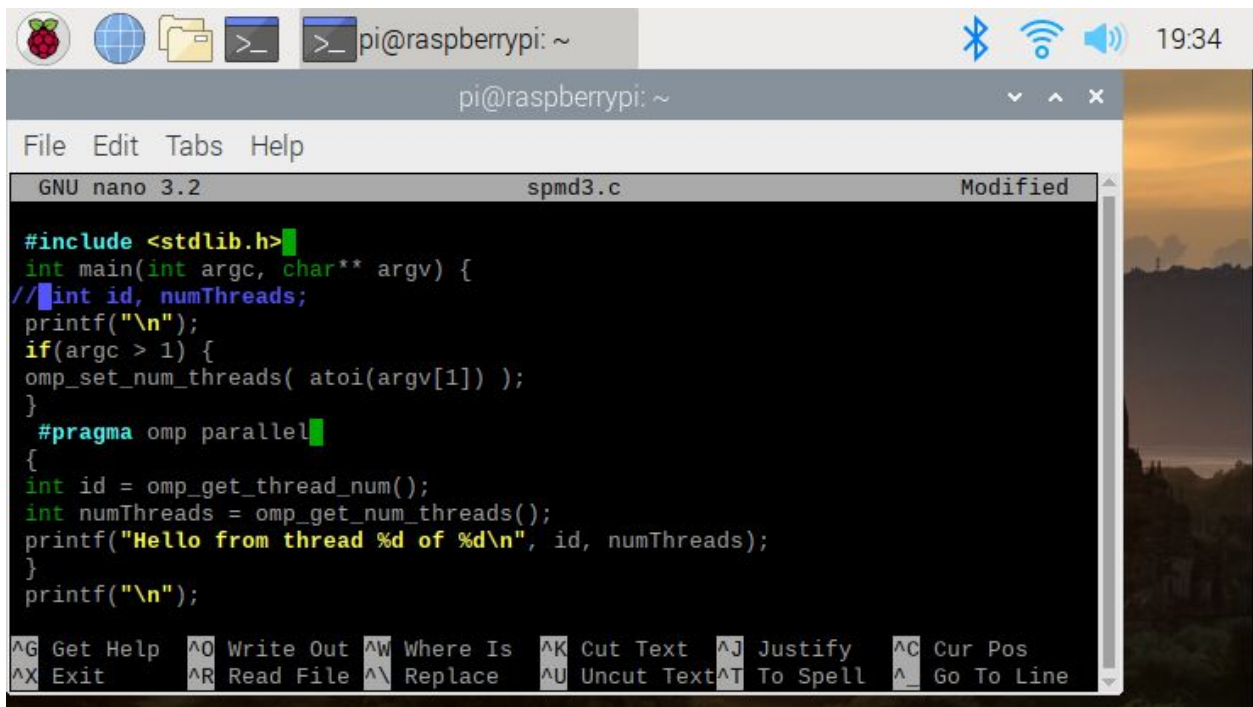
A terminal window on a Raspberry Pi. The window title is 'pi@raspberrypi: ~'. The menu bar shows 'File Edit Tabs Help'. The prompt is 'pi@raspberrypi:~'. The user enters 'nano spmd3.c' and then './spmd3 5'. The output shows five lines: 'Hello from thread 2 of 5', 'Hello from thread 1 of 5', 'Hello from thread 3 of 5', 'Hello from thread 0 of 5', and 'Hello from thread 4 of 5'. The user then enters './spmd3 6' and the output shows six lines: 'Hello from thread 1 of 6', 'Hello from thread 5 of 6', 'Hello from thread 2 of 6', 'Hello from thread 3 of 6', 'Hello from thread 4 of 6', and 'Hello from thread 0 of 6'. The prompt is now 'pi@raspberrypi:~\$'.

```
pi@raspberrypi:~$ nano spmd3.c
pi@raspberrypi:~$ ./spmd3 5
Hello from thread 2 of 5
Hello from thread 1 of 5
Hello from thread 3 of 5
Hello from thread 0 of 5
Hello from thread 4 of 5

pi@raspberrypi:~$ ./spmd3 6
Hello from thread 1 of 6
Hello from thread 5 of 6
Hello from thread 2 of 6
Hello from thread 3 of 6
Hello from thread 4 of 6
Hello from thread 0 of 6

pi@raspberrypi:~$
```

The original source code gives an output of thread 2, thread 3, thread 2, thread 2. This is reasonable because the source code is designed to assign the id variables which is why the values of the thread ids share.

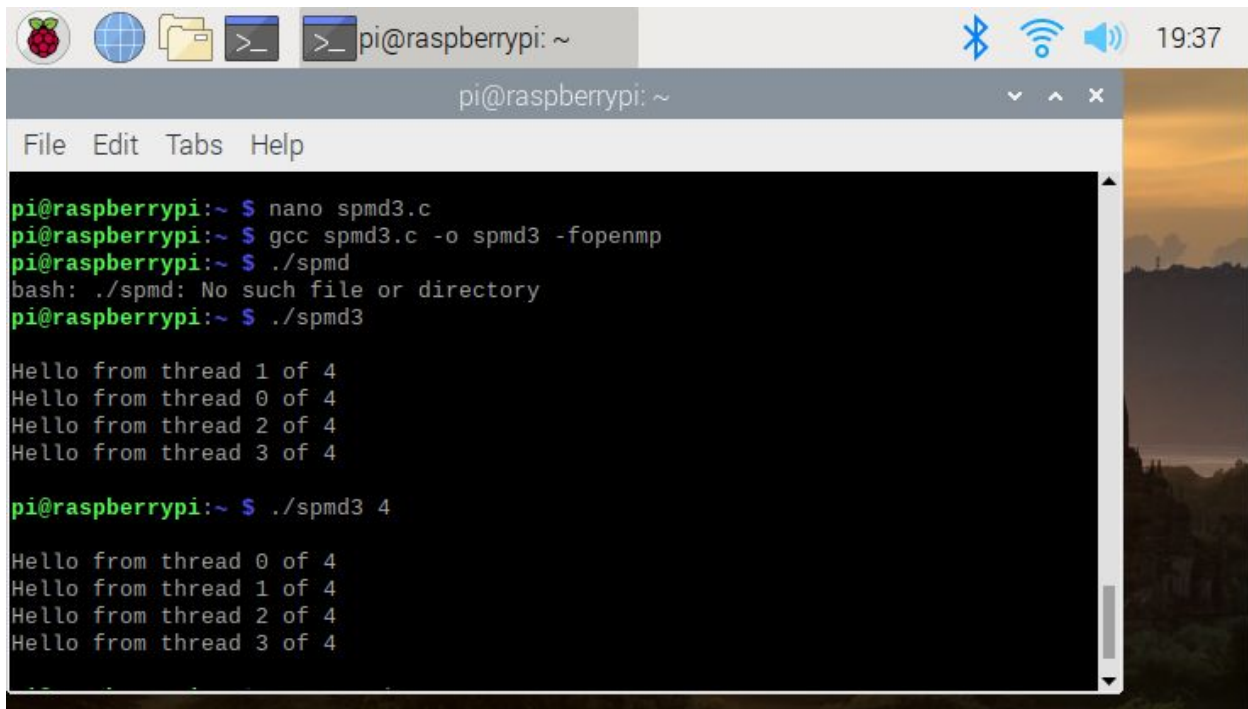


A terminal window showing the source code of 'spmd3.c' in the nano editor. The window title is 'pi@raspberrypi: ~'. The menu bar shows 'File Edit Tabs Help'. The status bar at the bottom shows 'GNU nano 3.2', 'spmd3.c', and 'Modified'. The code is as follows:

```
#include <stdlib.h>
int main(int argc, char** argv) {
//int id, numThreads;
printf("\n");
if(argc > 1) {
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");
}
```

The status bar at the bottom contains the following shortcuts: ^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.

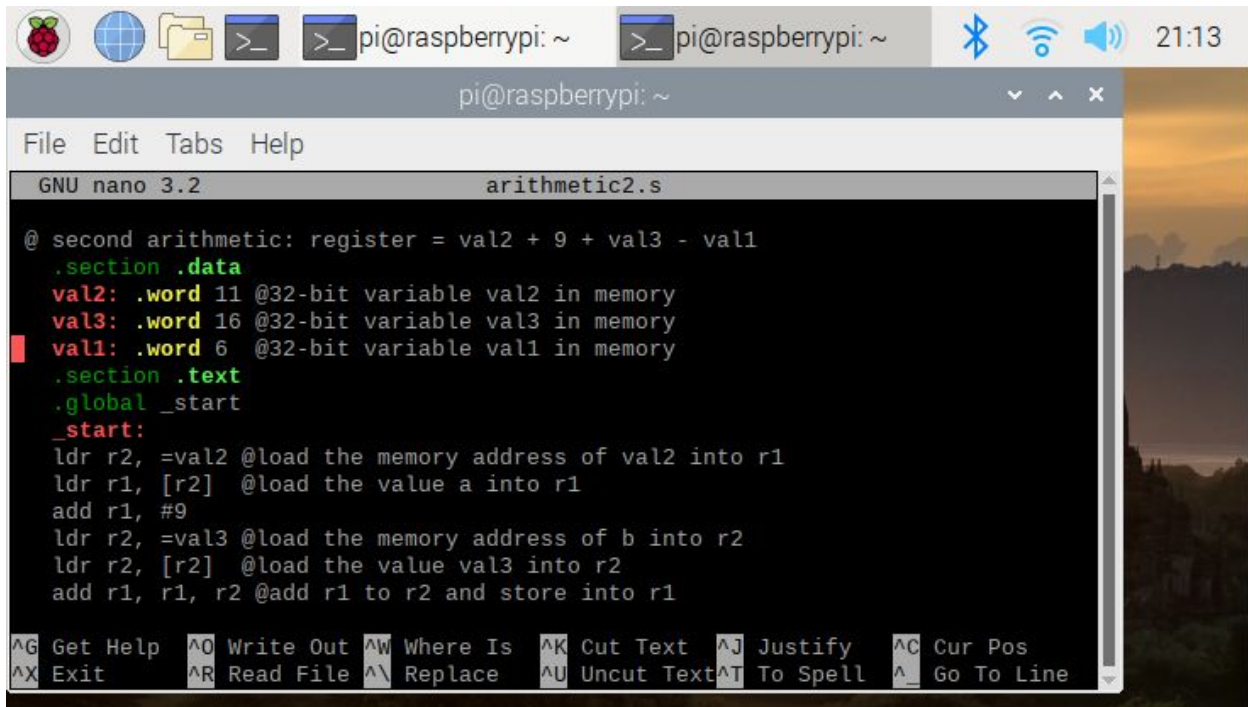
In order to make the threads align, the revised source code should comment the declaration of the id and numThreads (line 5). We individually declare the id and numThreads on lines 12 and 13.



```
pi@raspberrypi: ~  
File Edit Tabs Help  
pi@raspberrypi:~$ nano spmd3.c  
pi@raspberrypi:~$ gcc spmd3.c -o spmd3 -fopenmp  
pi@raspberrypi:~$ ./spmd  
bash: ./spmd: No such file or directory  
pi@raspberrypi:~$ ./spmd3  
  
Hello from thread 1 of 4  
Hello from thread 0 of 4  
Hello from thread 2 of 4  
Hello from thread 3 of 4  
  
pi@raspberrypi:~$ ./spmd3 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
```

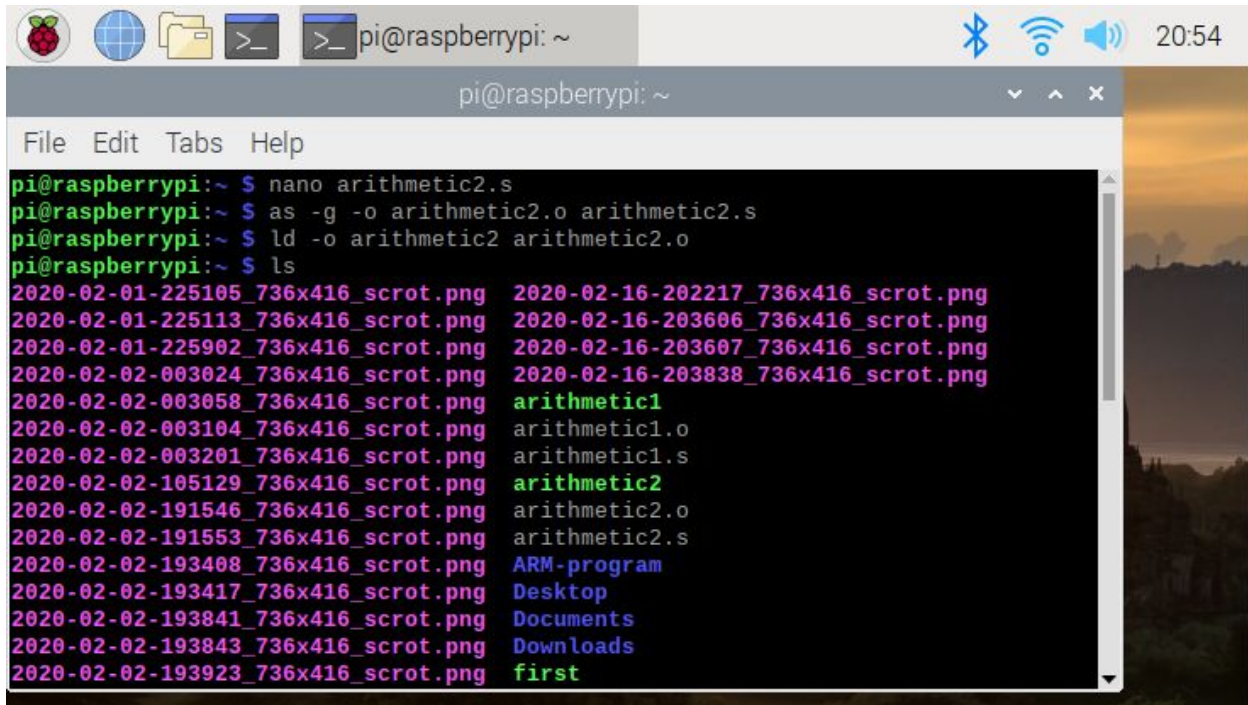
After editing the source code, we can see that the threads are aligned which was the goal where all 4 threads are shown and each have different ids and are aligned.

ARM PROGRAMMING



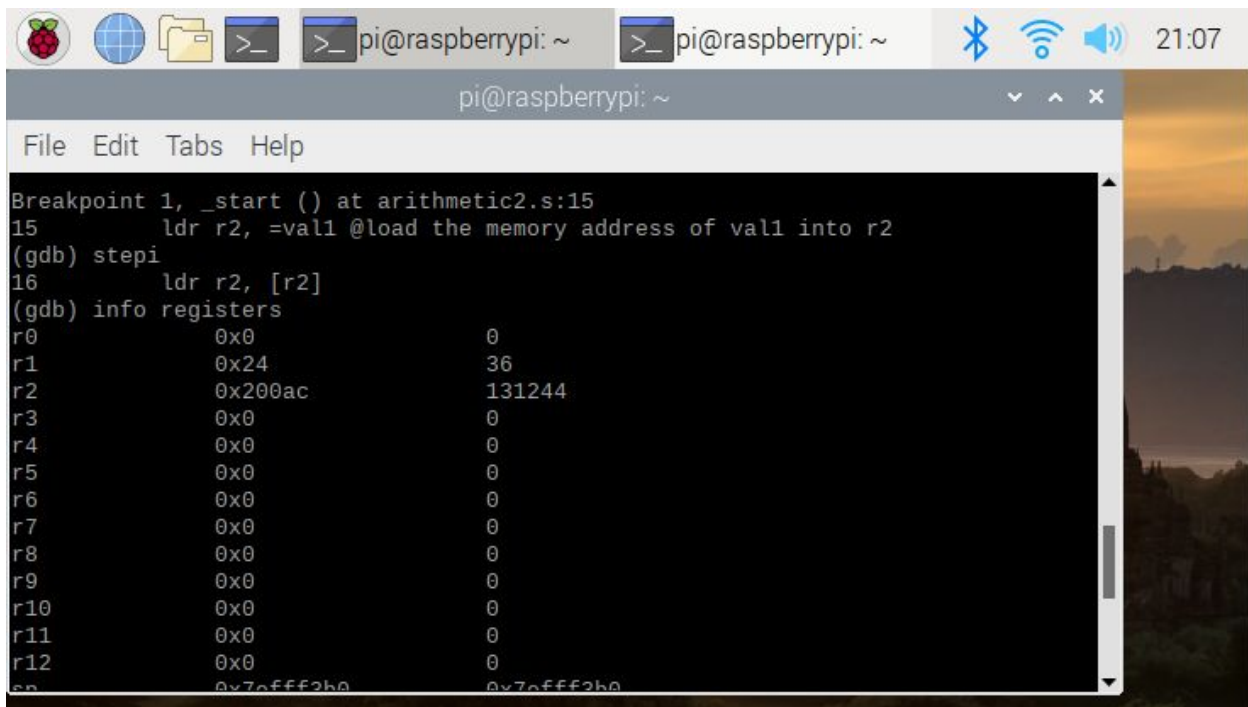
```
GNU nano 3.2 arithmetic2.s  
@ second arithmetic: register = val2 + 9 + val3 - val1  
.section .data  
val2: .word 11 @32-bit variable val2 in memory  
val3: .word 16 @32-bit variable val3 in memory  
val1: .word 6 @32-bit variable val1 in memory  
.section .text  
.global _start  
_start:  
ldr r2, =val2 @load the memory address of val2 into r1  
ldr r1, [r2] @load the value a into r1  
add r1, #9  
ldr r2, =val3 @load the memory address of b into r2  
ldr r2, [r2] @load the value val3 into r2  
add r1, r1, r2 @add r1 to r2 and store into r1  
  
^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 the source code for arithmetic2.s. In this code it can be seen that 11 is stored in val2, 16 in val3 and 6 in val1. The goal of this code is to print the result of $11 + 9 + 16 - 6$.



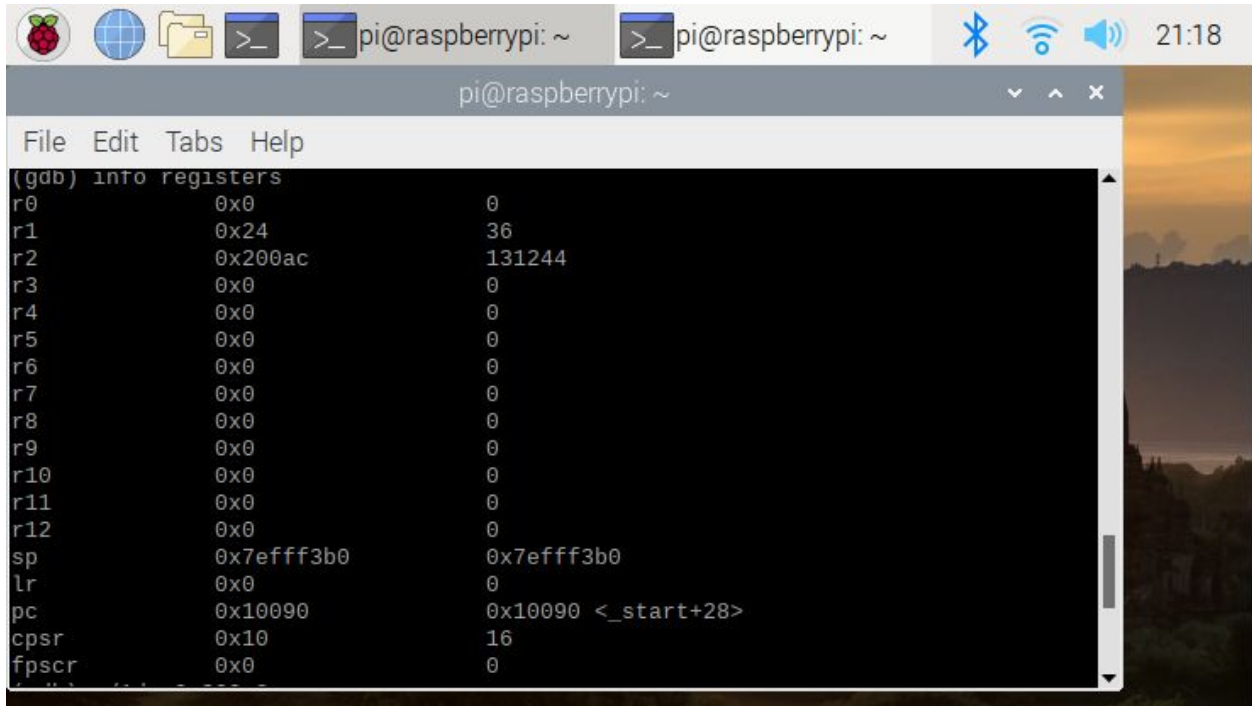
```
pi@raspberrypi:~  
File Edit Tabs Help  
pi@raspberrypi:~ $ nano arithmetic2.s  
pi@raspberrypi:~ $ as -g -o arithmetic2.o arithmetic2.s  
pi@raspberrypi:~ $ ld -o arithmetic2 arithmetic2.o  
pi@raspberrypi:~ $ ls  
2020-02-01-225105_736x416_scrot.png 2020-02-16-202217_736x416_scrot.png  
2020-02-01-225113_736x416_scrot.png 2020-02-16-203606_736x416_scrot.png  
2020-02-01-225902_736x416_scrot.png 2020-02-16-203607_736x416_scrot.png  
2020-02-02-003024_736x416_scrot.png 2020-02-16-203838_736x416_scrot.png  
2020-02-02-003058_736x416_scrot.png arithmetic1  
2020-02-02-003104_736x416_scrot.png arithmetic1.o  
2020-02-02-003201_736x416_scrot.png arithmetic1.s  
2020-02-02-105129_736x416_scrot.png arithmetic2  
2020-02-02-191546_736x416_scrot.png arithmetic2.o  
2020-02-02-191553_736x416_scrot.png arithmetic2.s  
2020-02-02-193408_736x416_scrot.png ARM-program  
2020-02-02-193417_736x416_scrot.png Desktop  
2020-02-02-193841_736x416_scrot.png Documents  
2020-02-02-193843_736x416_scrot.png Downloads  
2020-02-02-193923_736x416_scrot.png first
```

Command ls was used to check the current directories and to make sure the current file was saved in the places intended for it to be saved.



```
pi@raspberrypi:~  
File Edit Tabs Help  
Breakpoint 1, _start () at arithmetic2.s:15  
15      ldr r2, =val1 @load the memory address of val1 into r2  
(gdb) stepi  
16      ldr r2, [r2]  
(gdb) info registers  
r0          0x0          0  
r1          0x24         36  
r2          0x200ac       131244  
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          0x7cfff2b0     0x7cfff2b0
```

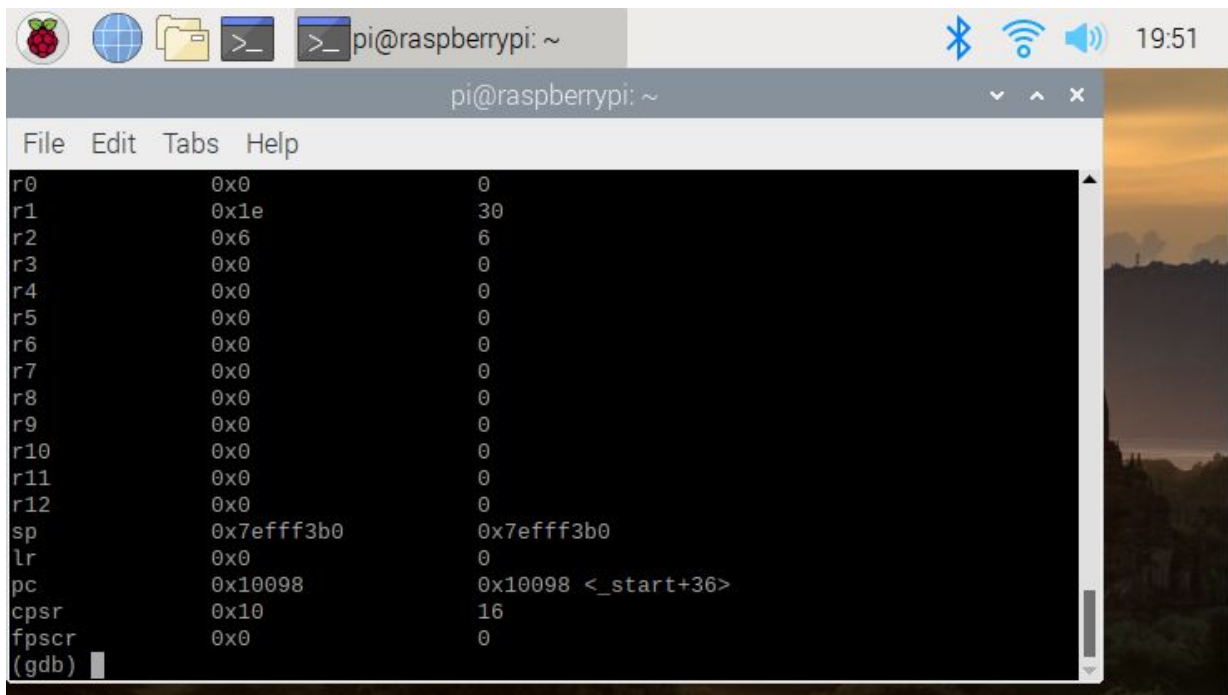
The breakpoint is set to line 15 and (gdb) info registers pulls up all the registers and the stored values in them. And based on this you can see that register r2 is the location where it is saved.



The screenshot shows a terminal window on a Raspberry Pi. The title bar indicates the user is at the home directory (~) on a pi@raspberrypi. The terminal content shows the command (gdb) info registers, which displays the current state of the CPU registers. The registers listed are r0 through r12, sp, lr, pc, cpsr, and fpscr. The values for r0 through r12 are mostly 0, except for r1 which is 36 and r2 which is 131244. The stack pointer (sp) is 0x7efff3b0. The program counter (pc) is 0x10090, which is labeled as <_start+28>. The current time is 21:18.

Register	Hex Value	Dec Value
r0	0x0	0
r1	0x24	36
r2	0x200ac	131244
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	0x7efff3b0	0x7efff3b0
lr	0x0	0
pc	0x10090	0x10090 <_start+28>
cpsr	0x10	16
fpscr	0x0	0

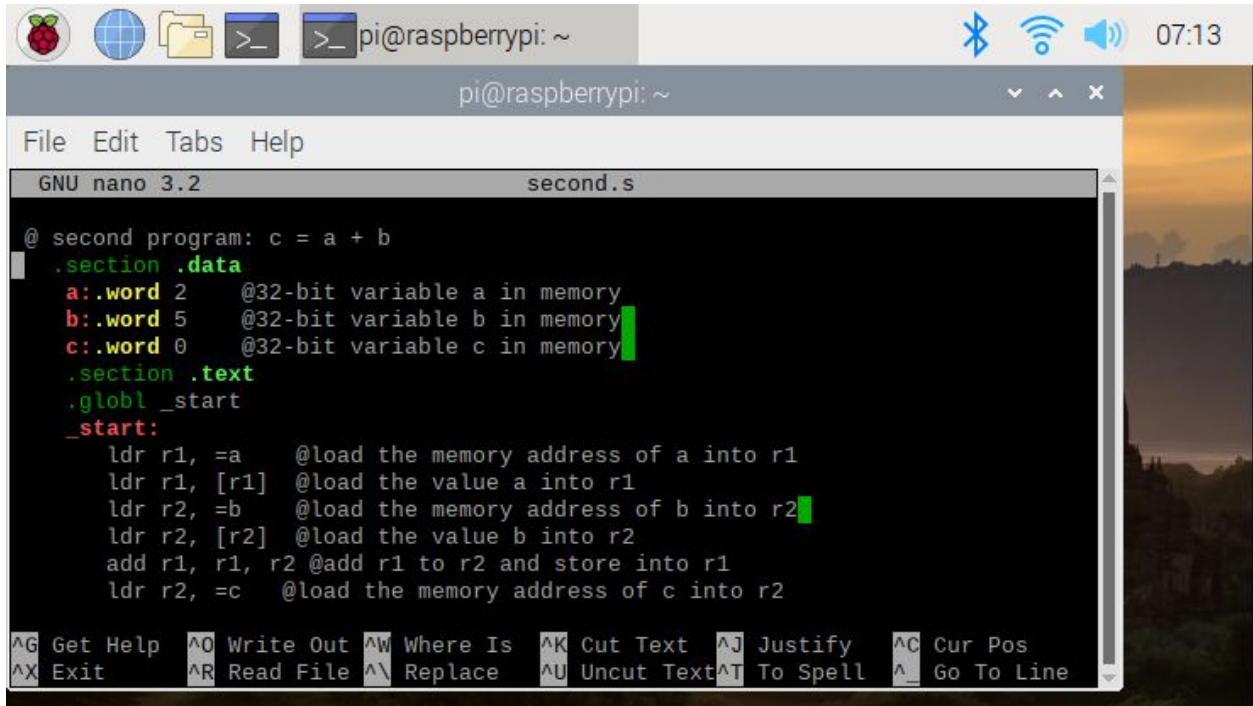
6 comes from val1 and adding 16 to 20 gives 36.



The screenshot shows the same terminal window after an operation. The (gdb) info registers command has been run again. The register values have changed: r1 is now 30 (hex 0x1e) and r2 is now 6 (hex 0x6). The program counter (pc) is now 0x10098, labeled as <_start+36>. The current time is 19:51.

Register	Hex Value	Dec Value
r0	0x0	0
r1	0x1e	30
r2	0x6	6
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	0x7efff3b0	0x7efff3b0
lr	0x0	0
pc	0x10098	0x10098 <_start+36>
cpsr	0x10	16
fpscr	0x0	0

30 is the final result after subtracting 6 from 36 and it is stored in r1. Since the output shows what is being expected, it is a valid output. And 0x1e shows the hexadecimal format of 30.

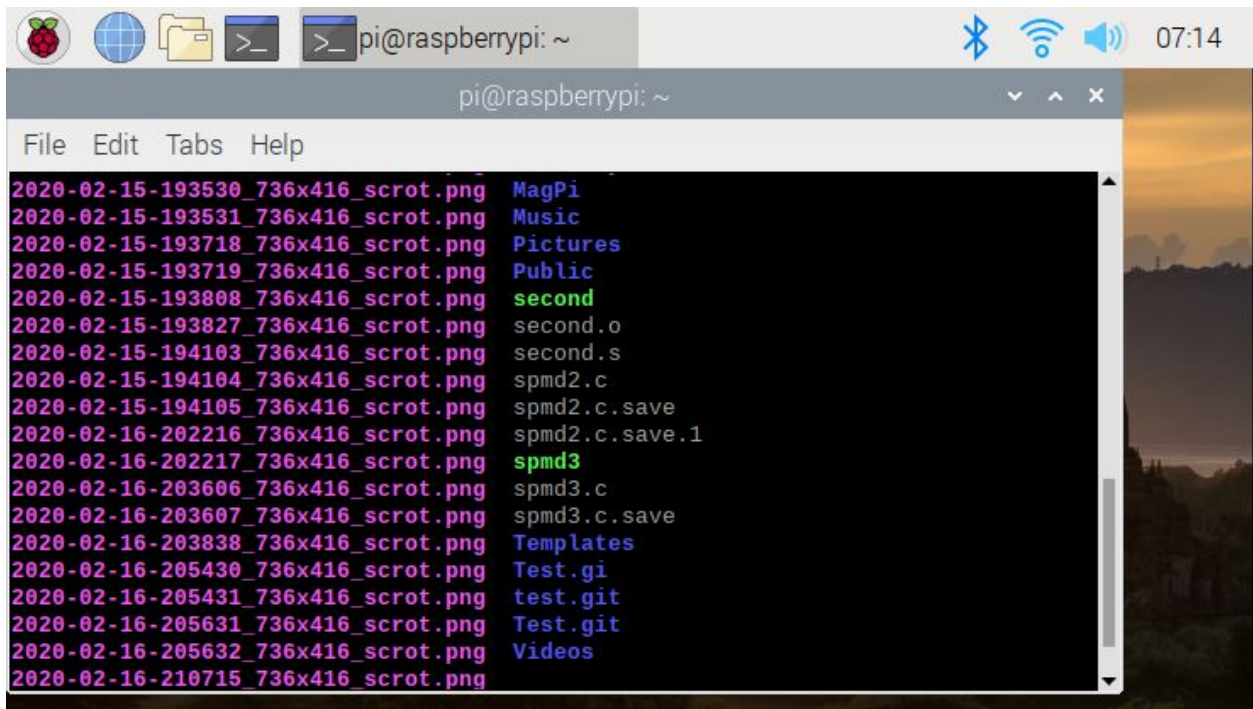


The screenshot shows a terminal window on a Raspberry Pi with the nano text editor open. The file being edited is 'second.s'. The code is assembly language for ARM, defining variables a, b, and c, and performing an addition. The terminal window has a title bar with 'pi@raspberrypi: ~' and standard window controls. The nano editor's status bar at the bottom shows various keyboard shortcuts.

```
GNU nano 3.2 second.s
@ second program: c = a + b
.section .data
a:.word 2 @32-bit variable a in memory
b:.word 5 @32-bit variable b in memory
c:.word 0 @32-bit variable c in memory
.section .text
.globl _start
_start:
    ldr r1, =a @load the memory address of a into r1
    ldr r1, [r1] @load the value a into r1
    ldr r2, =b @load the memory address of b into r2
    ldr r2, [r2] @load the value b into r2
    add r1, r1, r2 @add r1 to r2 and store into r1
    ldr r2, =c @load the memory address of c into r2

^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
```

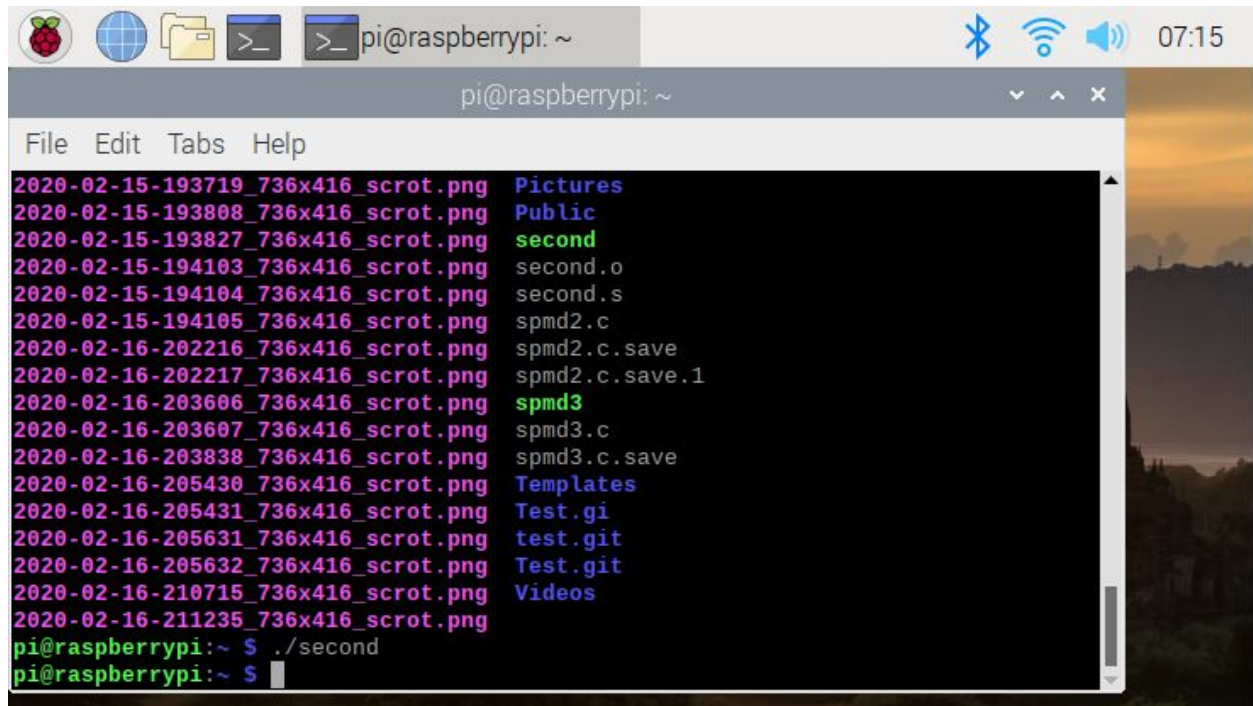
This is the source code for second.s. It traces the code that there is 2 word value in a memory address, 5 word value in the b memory address and 0 word value in the c memory address. After declaring this, the address of a will be loaded into register r1, therefore 2 will be stored in r1. The address of b will be loaded into register r2. Since there is an add function, 2 and 5 will be added and stored into r1. C is being loaded in r2 where 7 (2+5) is stored.



The screenshot shows a terminal window on a Raspberry Pi displaying a directory listing of files in the home directory. The files are listed with their full paths and names. The terminal window has a title bar with 'pi@raspberrypi: ~' and standard window controls.

```
2020-02-15-193530_736x416_scrot.png MagPi
2020-02-15-193531_736x416_scrot.png Music
2020-02-15-193718_736x416_scrot.png Pictures
2020-02-15-193719_736x416_scrot.png Public
2020-02-15-193808_736x416_scrot.png second
2020-02-15-193827_736x416_scrot.png second.o
2020-02-15-194103_736x416_scrot.png second.s
2020-02-15-194104_736x416_scrot.png spmd2.c
2020-02-15-194105_736x416_scrot.png spmd2.c.save
2020-02-16-202216_736x416_scrot.png spmd2.c.save.1
2020-02-16-202217_736x416_scrot.png spmd3
2020-02-16-203606_736x416_scrot.png spmd3.c
2020-02-16-203607_736x416_scrot.png spmd3.c.save
2020-02-16-203838_736x416_scrot.png Templates
2020-02-16-205430_736x416_scrot.png Test.gi
2020-02-16-205431_736x416_scrot.png test.git
2020-02-16-205631_736x416_scrot.png Test.git
2020-02-16-205632_736x416_scrot.png Videos
2020-02-16-210715_736x416_scrot.png
```

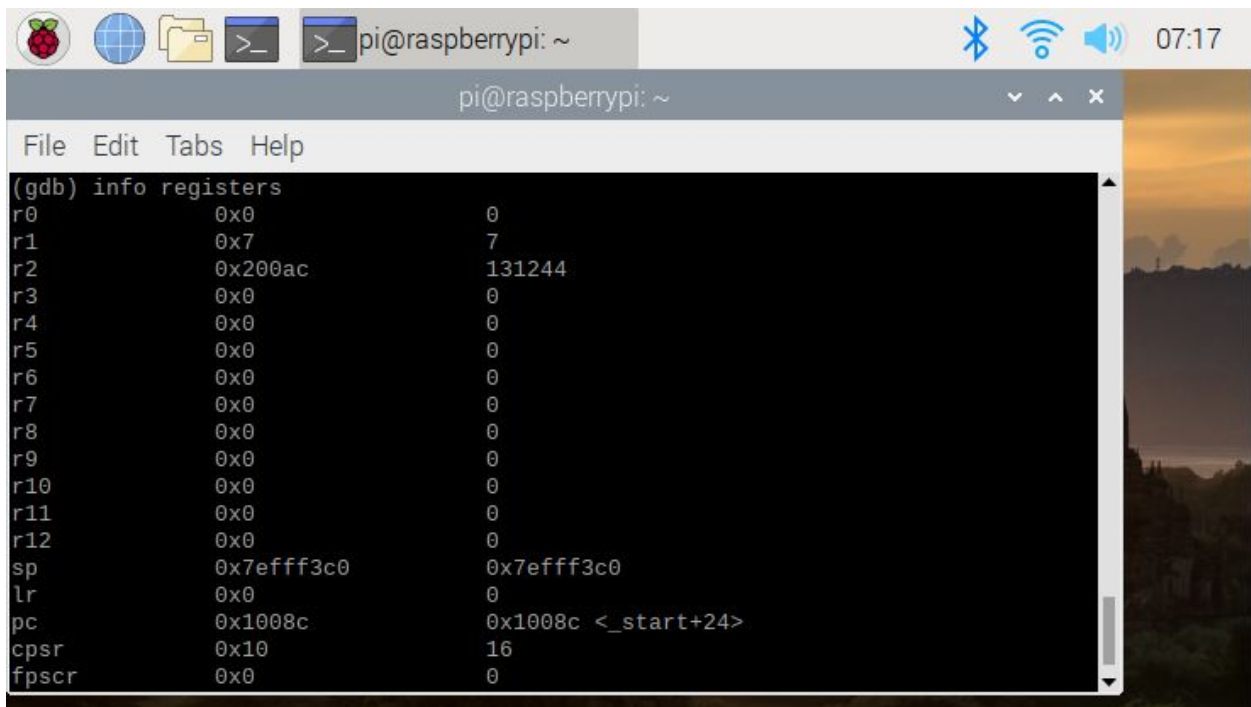

Command ls was used to check the current directories.



```
pi@raspberrypi: ~
File Edit Tabs Help
2020-02-15-193719_736x416_scrot.png Pictures
2020-02-15-193808_736x416_scrot.png Public
2020-02-15-193827_736x416_scrot.png second
2020-02-15-194103_736x416_scrot.png second.o
2020-02-15-194104_736x416_scrot.png second.s
2020-02-15-194105_736x416_scrot.png spmd2.c
2020-02-16-202216_736x416_scrot.png spmd2.c.save
2020-02-16-202217_736x416_scrot.png spmd2.c.save.1
2020-02-16-203606_736x416_scrot.png spmd3
2020-02-16-203607_736x416_scrot.png spmd3.c
2020-02-16-203838_736x416_scrot.png spmd3.c.save
2020-02-16-205430_736x416_scrot.png Templates
2020-02-16-205431_736x416_scrot.png Test.git
2020-02-16-205631_736x416_scrot.png test.git
2020-02-16-205632_736x416_scrot.png Test.git
2020-02-16-210715_736x416_scrot.png Videos
2020-02-16-211235_736x416_scrot.png
pi@raspberrypi:~ $ ./second
pi@raspberrypi:~ $
```

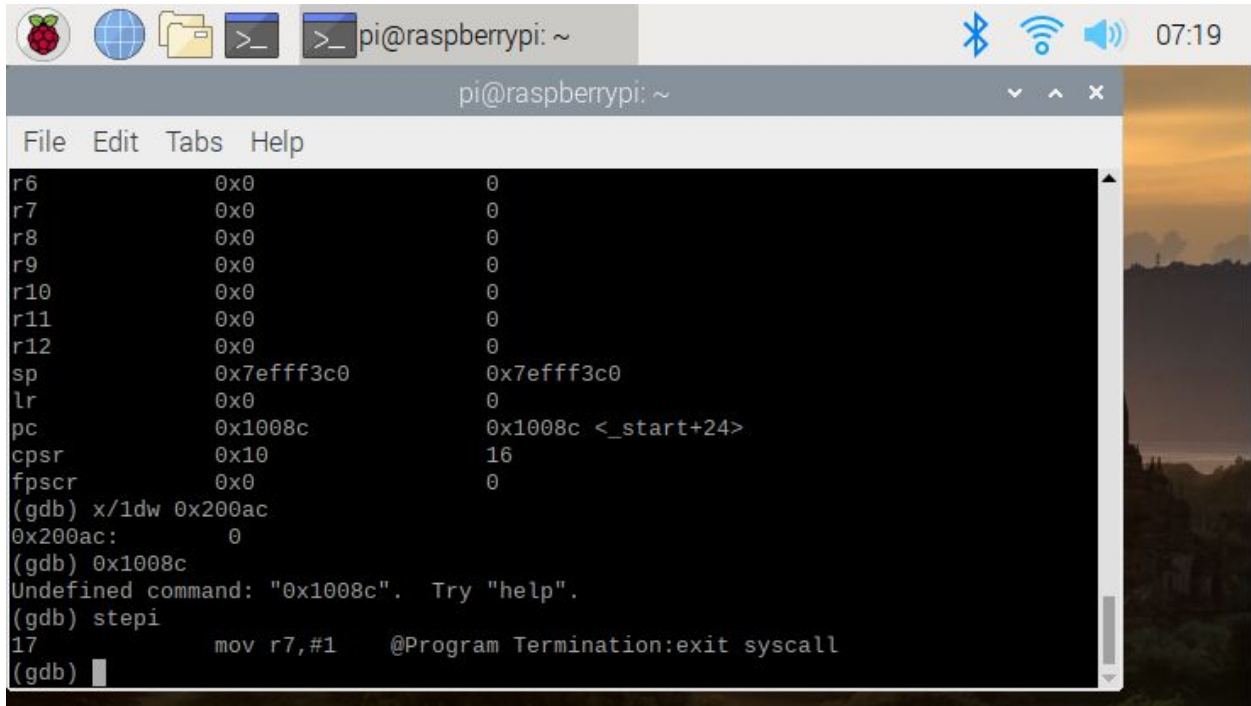
There was no output after entering `./second` because the code in `second.s` only manipulates the numbers in the registers and does not print or return any values. Therefore, there is no output.

vfr4y



```
pi@raspberrypi: ~
File Edit Tabs Help
(gdb) info registers
r0          0x0          0
r1          0x7          7
r2          0x200ac      131244
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          0x7efff3c0    0x7efff3c0
lr          0x0          0
pc          0x1008c      0x1008c <_start+24>
cpsr       0x10         16
fpscr      0x0          0
```

(gdb) info registers lets us know what is stored in each register.

A screenshot of a Raspberry Pi terminal window. The window title is 'pi@raspberrypi: ~'. The terminal shows a GDB session. It starts with a register dump for registers r6 through r12, sp, lr, pc, cpsr, and fpscr. The values for r6-r12 are 0x0. sp is 0x7efff3c0. lr is 0x0. pc is 0x1008c. cpsr is 0x10. fpscr is 0x0. Then, the user enters '(gdb) x/1dw 0x200ac', and the output is '0x200ac: 0'. Next, the user enters '(gdb) 0x1008c', and the output is 'Undefined command: "0x1008c". Try "help".'. Then, the user enters '(gdb) stepi', and the output is 'i7 mov r7,#1 @Program Termination:exit syscall'. Finally, the user enters '(gdb)'. The terminal window has a menu bar with 'File', 'Edit', 'Tabs', and 'Help'. The background of the terminal window shows a sunset over a landscape.

x/ 1 was used to look at the memory in only one entry. dw indicated the decimal format in a word size entry. 0x200ac that is stored at r2 indicates that the memory location has been stored at this point and its 0. Step i is used to run the last line of the code.

Parallel Programming Questions

1. Identifying the components on the raspberry PI B+.

The components include board computer, 1 gigabyte RAM and multi-core CPI

2. How many cores does the Raspberry Pi's B+ CPU have?

The Raspberry Pi's B+ CPU has four cores.

3. List three main differences between X86 (CISC) and ARM Raspberry PI (RISC). Justify your answer and use your own words (do not copy and paste).

ARM are RISC processors where the instructions are only in the form of registers unlike CISC, RISC has a more generalized form of registers. CISC has more instructions to accomplish a task than RISC, therefore the code is more complex in CISC systems. Latest ARM processors are using bi-endian instead of little endian and be eligible for changing endianness according to the code.

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

Parallel programming takes a problem and breaks it more and more so that it can be executed all at once by multiple processors. Sequential programming breaks a problem step by step and executes on one processor. The significance of this programming is that

when there is a problem that cannot be broken down into smaller independent instructions. The significance of parallel programming is that it uses the resources efficiently therefore it saves time.

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

Data parallelism is when the same operation is conducted on all the data presented and this can only be done when the instructions are independent. When there is more data to be covered, data parallelism is the best way because everything is done parallelly so it consumes less time. Task parallelism is what the computer performs on the data and not what the output of the data is. This is used only when the tasks are independent from each other.

6. Explain the differences between processes and threads.

The Process is when a program is being executed. Processes don't share memory spaces, instead they run separately in the memory spaces. Whereas threads share memory spaces. Thread is within the process that can be called for execution. Processes provide resources for the code to be executed. Processes deal only with the environment the code is written but a thread interacts with the hardware.

7. What is OpenMP and what is OpenMP pragmas?

OpenMP is an API that helps shared memory in multi platform programming on different operating systems. OpenMP pragmas is when a set of compilers controls how the program is executed. Pragmas are created in a way that if the compiler does not support them then the program will still execute a valid output without parallelism involved.

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

The top applications that benefit from multi-core are Web servers, compilers, database servers and Multimedia applications.

9. Why Multicore? (why not single core, list four)

- Since computer architecture follows the concept of parallel programming, it needs many threads instead of one.
- A single core requires expensive materials to be used and it needs high maintenance since they have intense pipelined circuits.
- The single threading has a limit to how fast it can be done. So anything that exceeds the point should use multiple threads for execution.
- Most upcoming applications are in the use of multithreading.

