Developing Soft and Parallel Programming Skills using Project-Based Learning

Spring 2020

The Commuters
Alaya Shack, Miguel Romo, Arteen Ghafourikia, Andre Nguyenphuc, Joan Galicia

Planning and Scheduling:

Assignee Name	Email	Task	Duration (hours)	Dependency	Due Date	Note
Alaya Shack	ashack1@student.gsu.edu	Create planning and scheduling table. Complete individual parallel programming skills task and ARM assembly programming task.	hours	(none)	02/19	Review report for grammatical/ spelling errors.
Miguel Romo (Coordinator)	mromo1@student.gsu.edu	Edit the video and include the link in the report.Complete individual parallel programming skills task and ARM assembly programming task.	hours	(none)	02/19	Review report for grammatical/ spelling errors.
Joan Galicia	jgalicia2@student.gsu.edu	Serve as the facilitator.Complete individual parallel programming skills task and ARM assembly programming task.	hours	(none)	02/19	Review report for grammatical/ spelling errors.

Arteen Ghafourikia	aghafourikia1@student.gsu.edu	Identify new To-do, In-progress, and Completed columns in Github. Get the report formatted correctly (fonts, page numbers, and sections).Complete individual parallel programming skills task and ARM assembly programming task.	hours	Github, each member's report, and the video.	02/20	24 hours before the due date, please have the report ready for all team members to review.
Andre Nguyenphuc	anguyenphuc1@student.gsu.edu	Send an invitation to teaching assistant through slack. Complete individual parallel programming skills task and ARM assembly programming task.	hours	(none)	02/19	Review report for grammatical/ spelling errors.

<u>Parallel Programming Skills Foundation: Alaya Shack</u>

- **Identifying the components of the raspberry PI B+.**
  - The components of the raspberry PI B+ include a single board computer with a quad-core multicore CPU that has 1GB of RAM, two USB ports, an ethernet port, an ethernet controller, two power sources, a HDMI port, a camera, and display.
- **How many cores does the Raspberry Pi's B+ CPU have?**

- ◆ The Raspberry Pi’s B+ CPU is quad core, which means it has four cores.

➔ **List three main differences between X86 (CISC) and ARM Raspberry PI (RISC). Justify your answer and use your own words.**

- ◆
- ◆
- ◆

➔ **What is the difference between sequential and parallel computation and identify the practical significance of each?**

- ◆

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

- ◆

➔ **Explain the differences between processes and threads.**

- ◆ One main difference between pr

➔ **What is OpenMP and what is OpenMP pragmas?**

- ◆

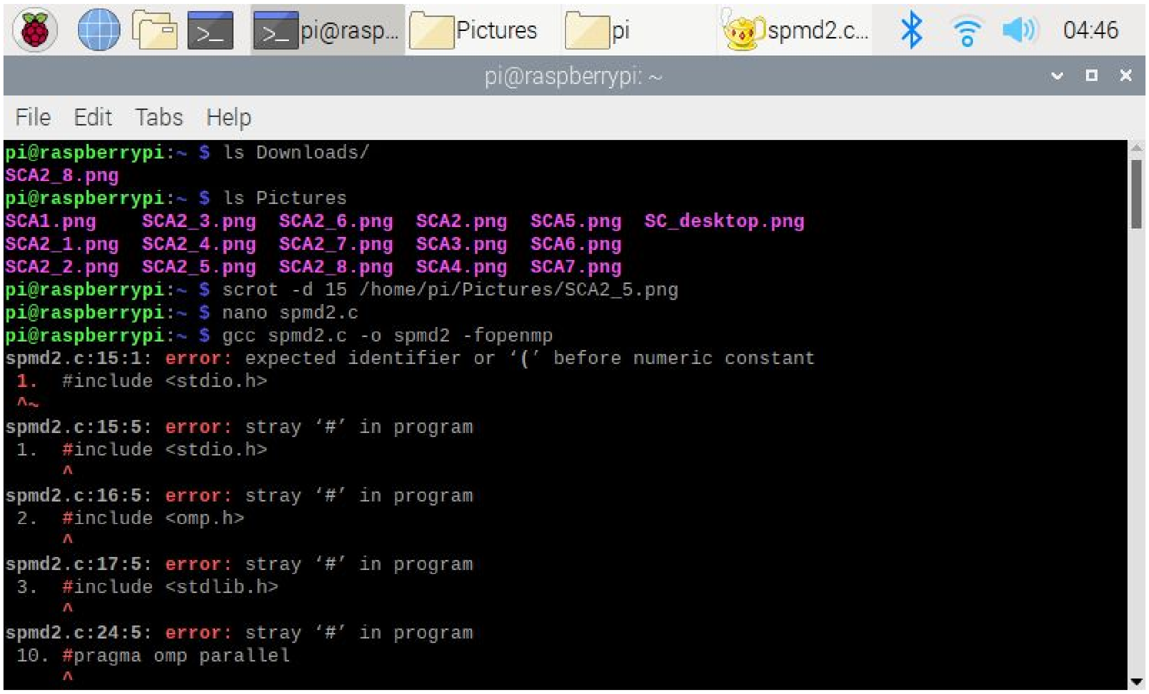
➔ **What applications benefit from multi-core (list four)?**

- ◆ Some applications that benefit from multi-core are:
  - Database servers
  - Compilers
  - Multimedia applications
  - Scientific applications such as CAD/CAM.

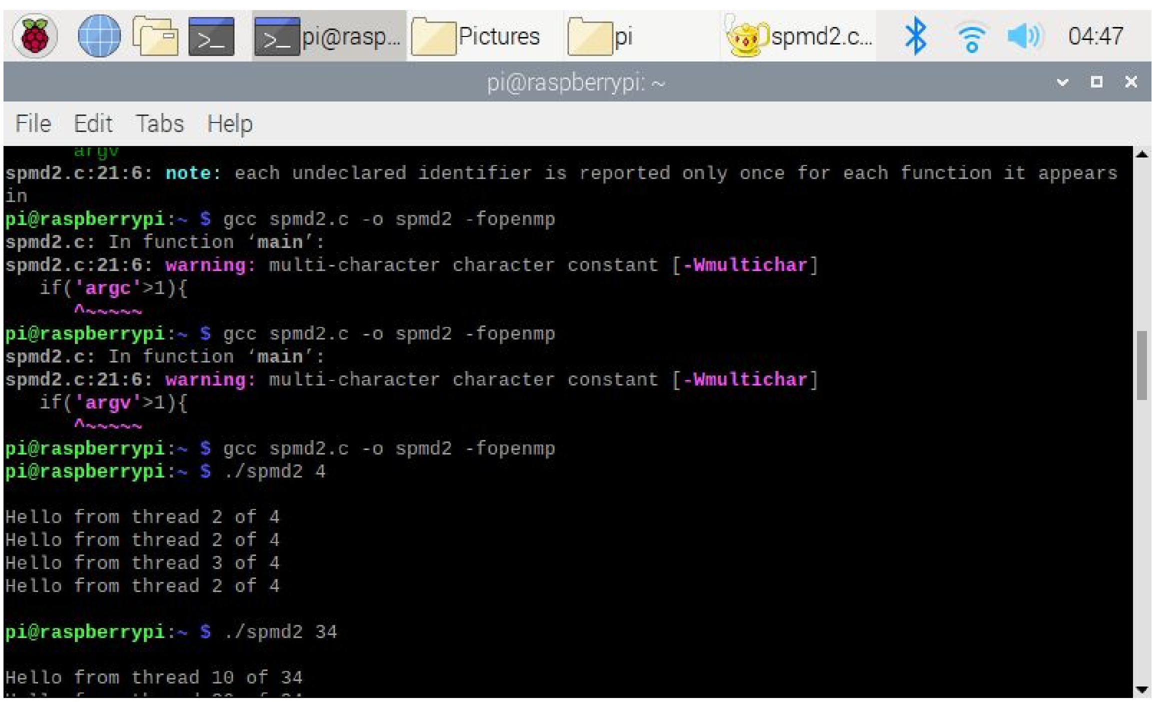
➔ **Why Multicore? (why not single core, list four)**

- ◆ These are some of the reasons why multi-core is preferred over single-core:
  - It is difficult to make single-core clock frequencies higher.
  - There is a general trend in computer architecture toward more parallelism.
  - Many new applications are multi-threaded

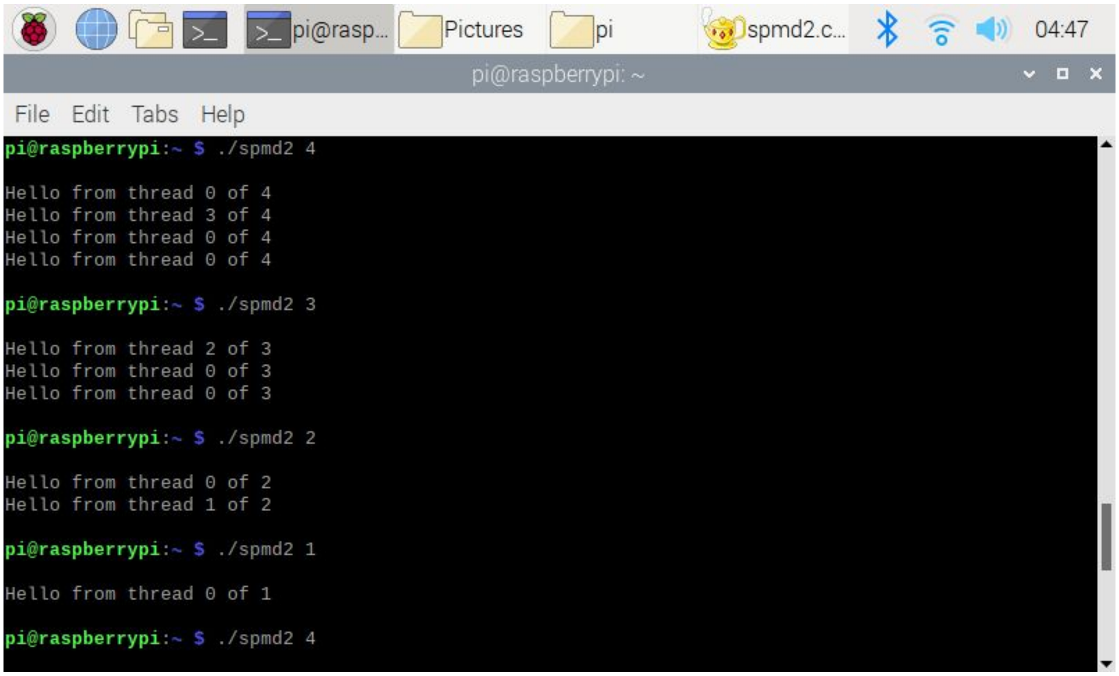
Parallel Programming Basics: Alaya Shack



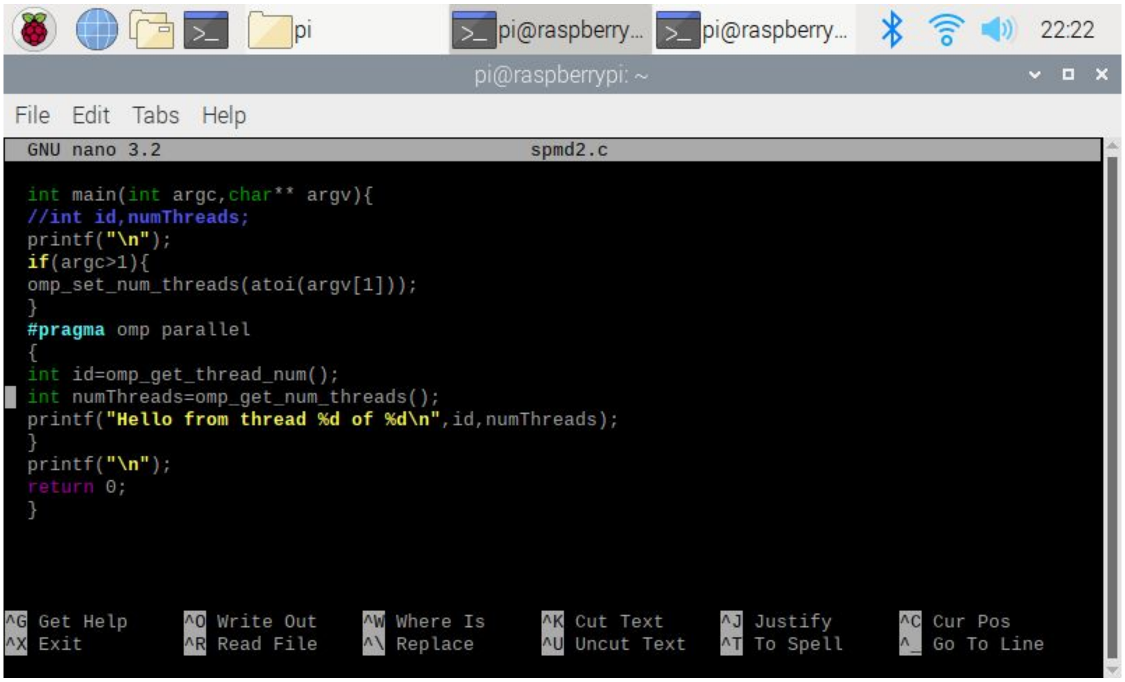
This screenshot is of using the terminal trick, tab completion of long names. I tried the trick with ls Dow[Tab], and I tried it with ls Pic[Tab]. It filled in the remaining part of the directories, and it listed all the items that belong in that are in that folder. Also, in this screenshot, I used "gcc spmd2.c -o spmd2 -fopenmp" to make the program executable, but I had errors.



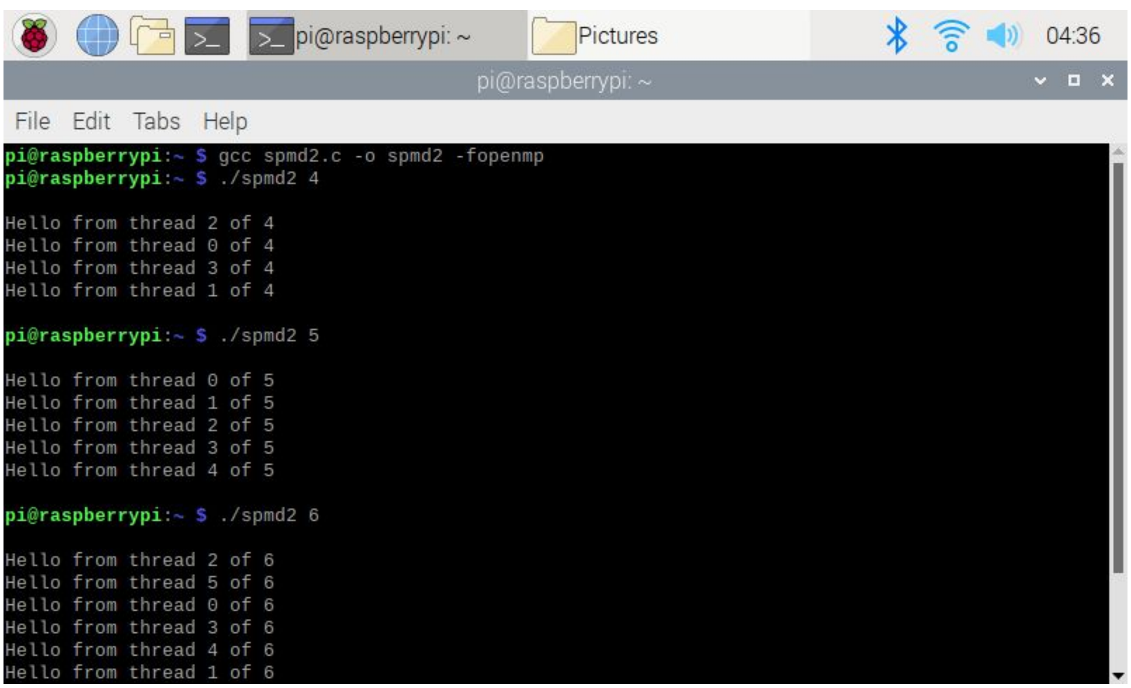
In this screenshot, I typed the same command to make the program executable. After I fixed my errors, I was finally able to make the spmd2.c an executable program. Then, I typed "./spmd2 4" to run the program.



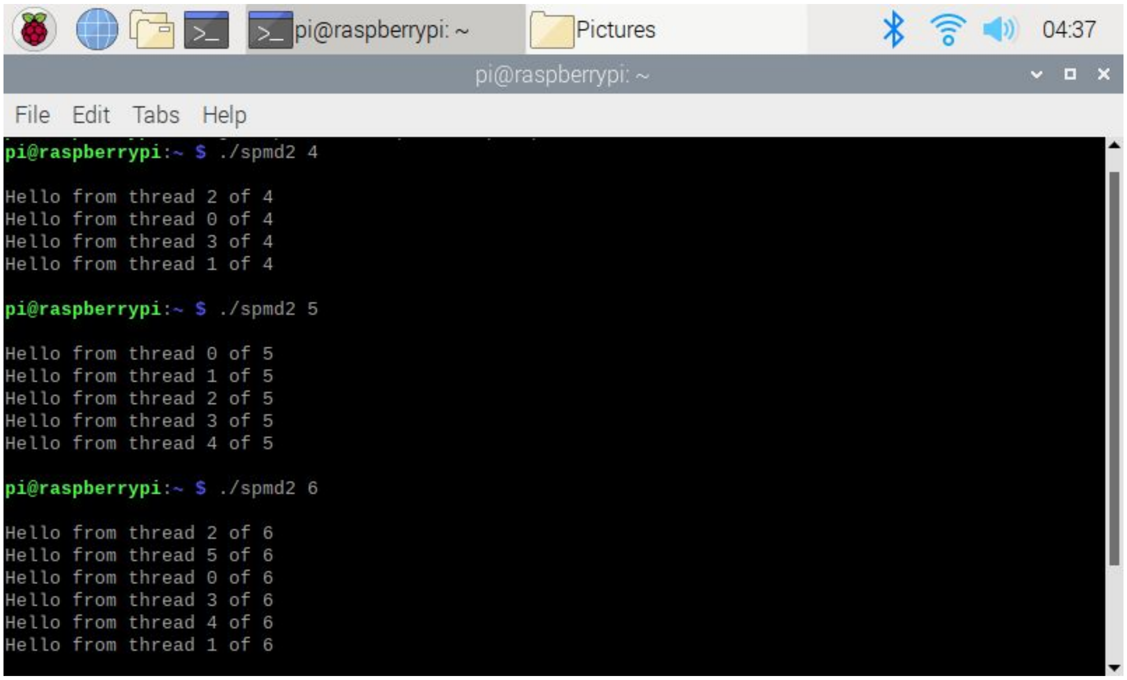
In this screenshot, I kept typing "./spmd2 4", but I alternated the 4 with 3,2, and 1. I learned that the 4 is a command-line argument that indicates how many threads to fork and that it can be changed. When I changed the argument, I noticed that the message printed the amount of times that the argument was. Also, I noticed that some of the messages would print the thread id multiple times and that the numbers do not always print out in order.



This screenshot is of the adjustments that were made to make the program run properly. First, I had to make line 5 a comment by placing two backslashes at the front of the line. Also, for lines 12 and 13, I had to properly declare the variable by placing int in front of the lines because the variables are of type integer.

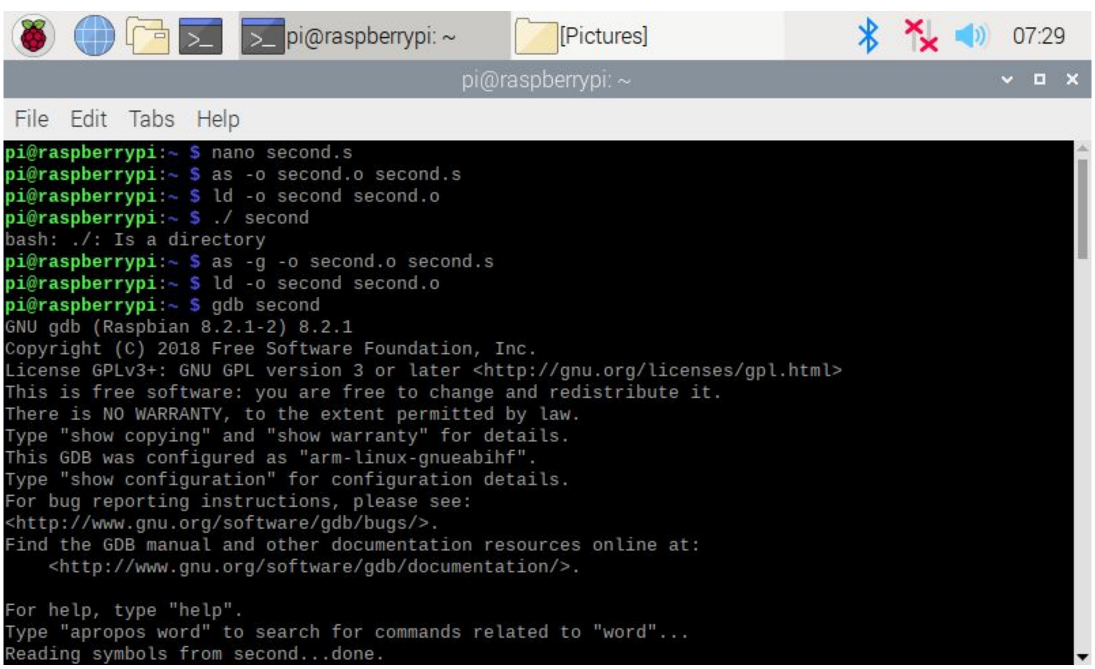


In this screenshot, I had made the changes to the code. I typed “gcc spmd2.c -o spmd2 -fopenmp” to compile the program, and I ran the program with “./spmd2 4”.

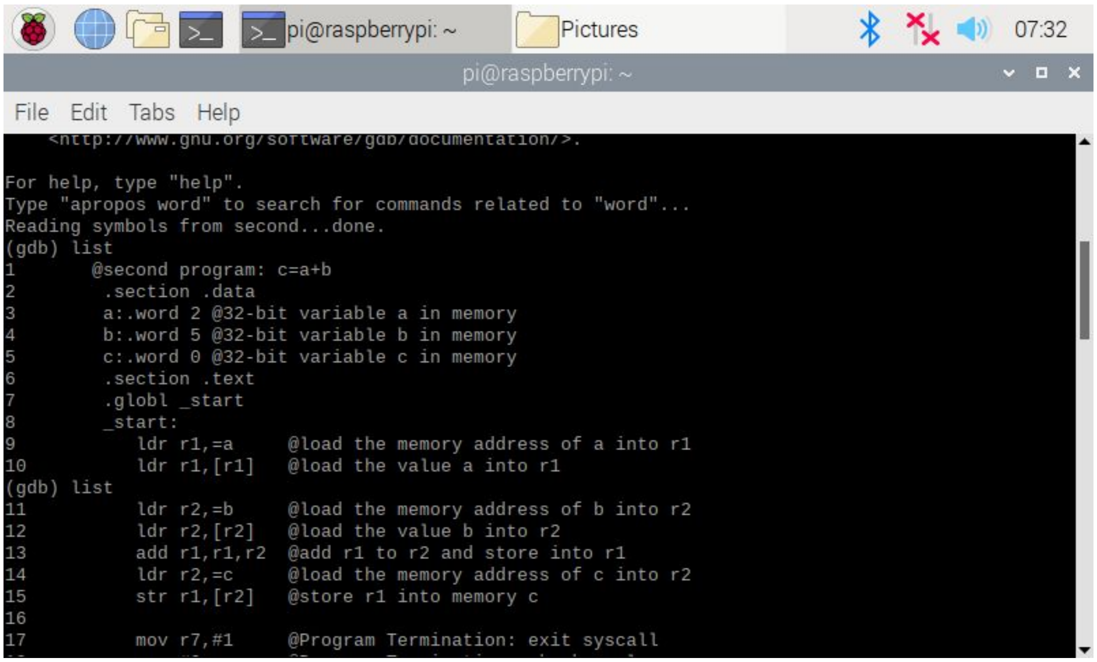


In this screenshot, I alternated the command-line argument 4 with 5 and 6 to make sure that the program was running correctly. The program was running correctly because now the thread id number only appears once. Although the thread id numbers do not print in order, I learned that this is okay because we do not know when a thread will finish.

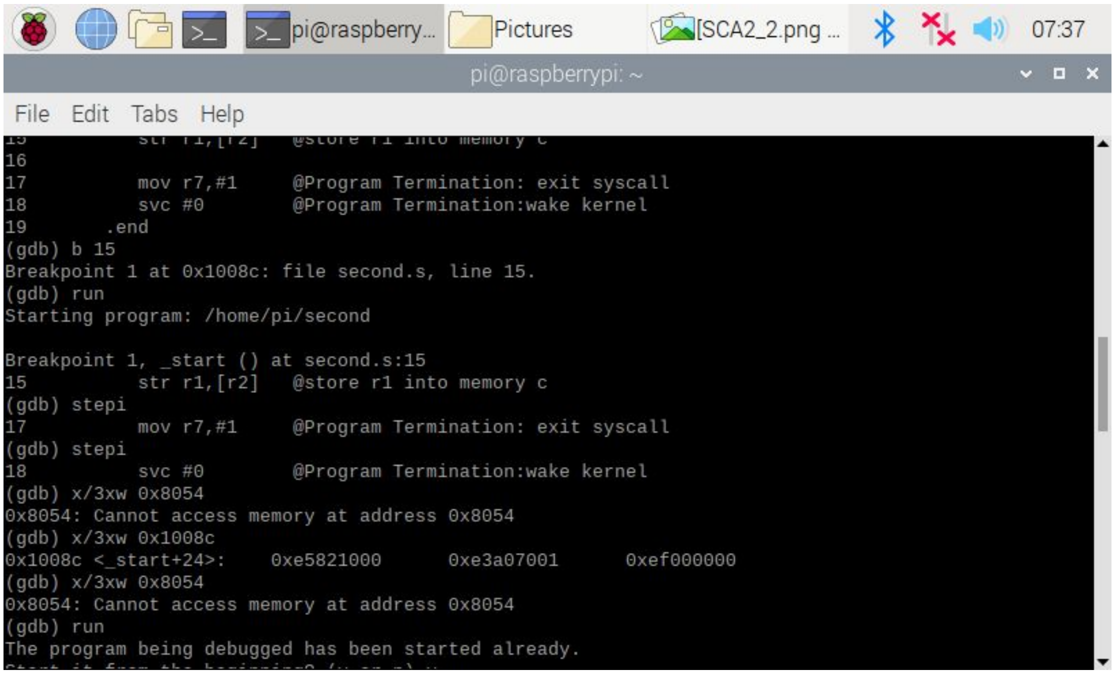
<u>Arm Assembly Programming:Alaya Shack</u>



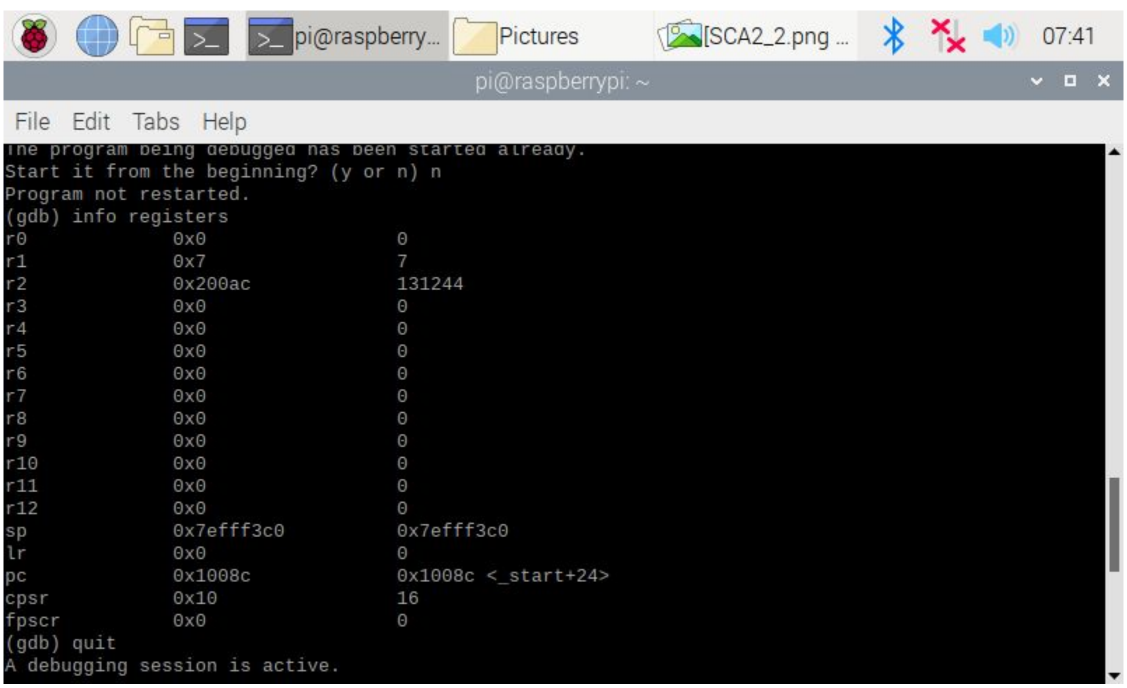
This screenshot shows how I started the “second” file with “nano second.s”. Once, I finished writing the file I was able to assemble and link the second program. I typed “./ second” to run the program, as directed by the assignment, and the “bash: ./: Is a directory” message appeared. However, I tried the “./second” and I did not get any output, which is what I expected. The second time that I assembled and linked the program, I added the “-g” flag for debugging. Then, I typed the “gdb second” command to launch the debugger.

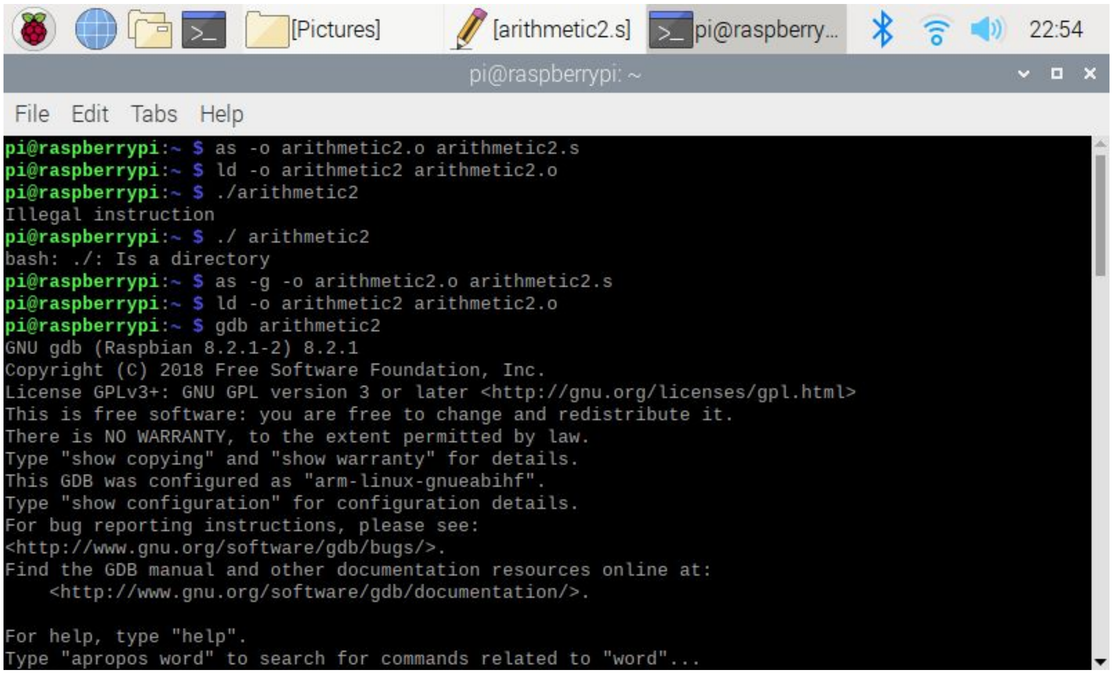


In this screenshot, I used the “gdb list” command to list the first ten lines of code, and I repeated the command and listed the next ten lines of code for the second program.



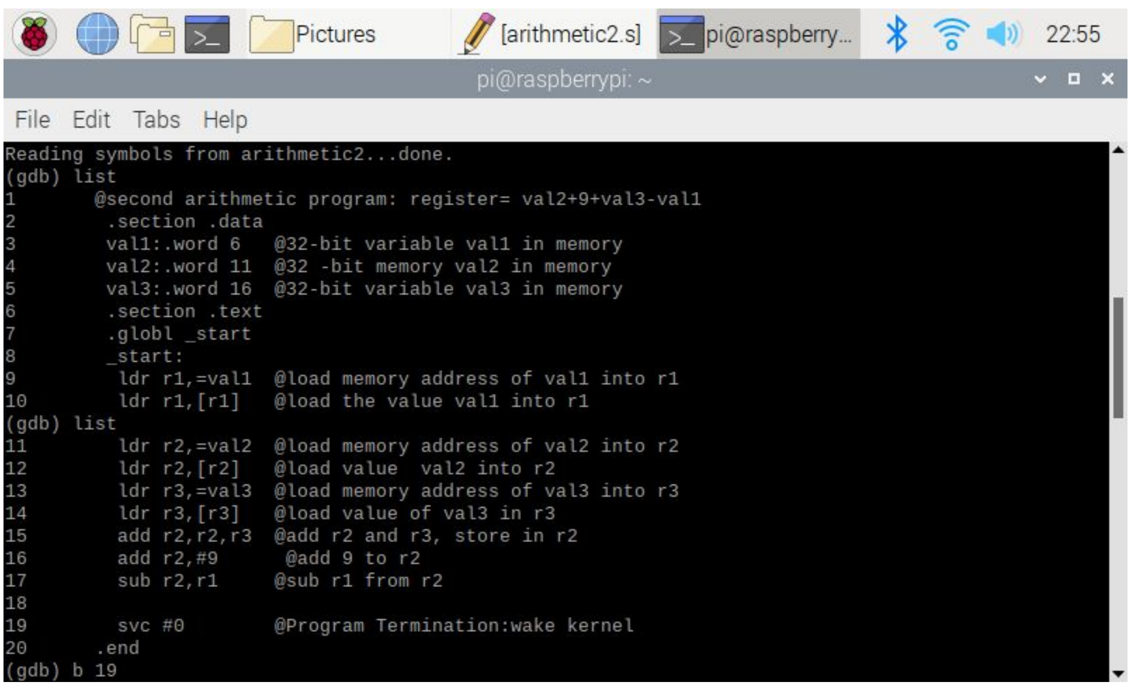
This screenshot shows where I inserted a breakpoint at line 15. After I inserted the breakpoint, I noticed that the next line shows the memory address of where I inserted the breakpoint. Also, in this screenshot, I started the debugging process with "gdb run". Then, I used "gdb stepi" to step through the program one line at a time. Next, I begin to examine the memory. At first, I copied the command verbatim from the ARM Assembly programming document, but I realized that I needed to use the memory address of where the breakpoint was inserted.



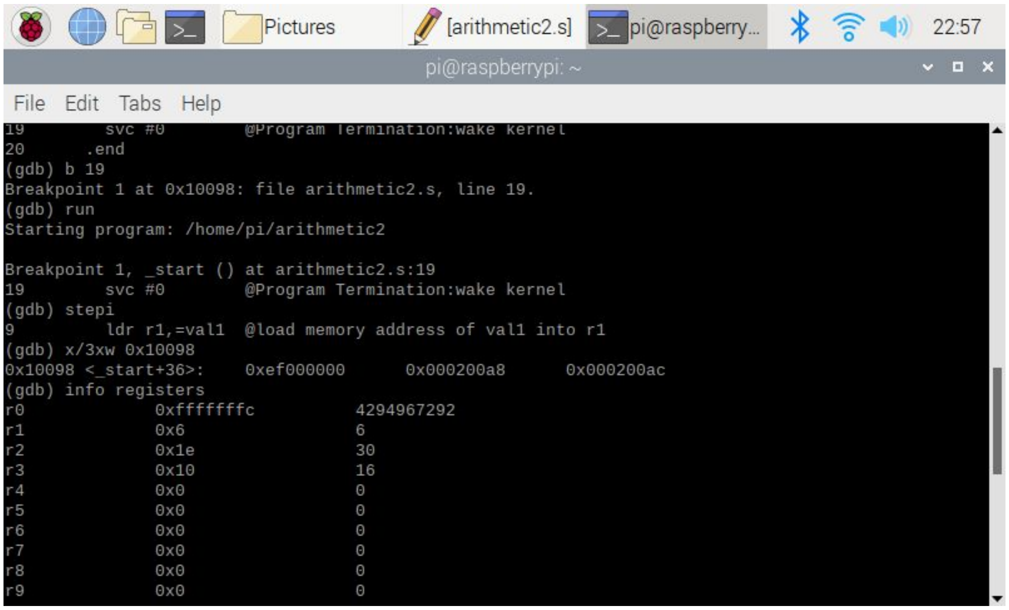


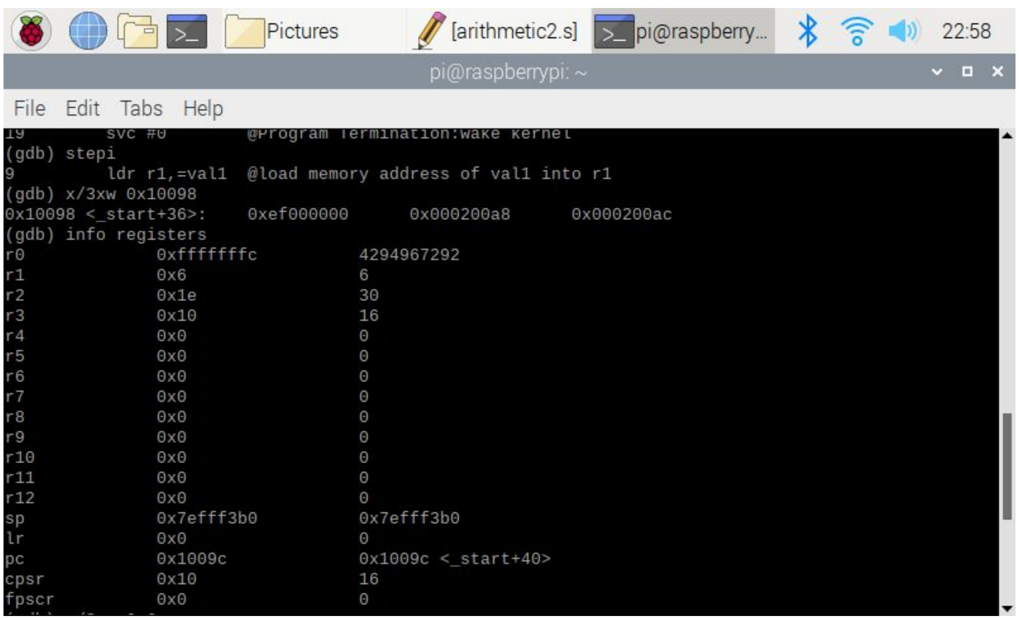

```
pi@raspberrypi:~ $ as -o arithmetic2.o arithmetic2.s
pi@raspberrypi:~ $ ld -o arithmetic2 arithmetic2.o
pi@raspberrypi:~ $ ./arithmetic2
Illegal instruction
pi@raspberrypi:~ $ ./ arithmetic2
bash: ./: Is a directory
pi@raspberrypi:~ $ as -g -o arithmetic2.o arithmetic2.s
pi@raspberrypi:~ $ ld -o arithmetic2 arithmetic2.o
pi@raspberrypi:~ $ gdb arithmetic2
GNU gdb (Raspbian 8.2.1-2) 8.2.1
Copyright (C) 2018 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-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 arithmetic2...done.
(gdb) list
1       @second arithmetic program: register= val2+9+val3-val1
2        .section .data
3        val1:.word 6   @32-bit variable val1 in memory
4        val2:.word 11  @32 -bit memory val2 in memory
5        val3:.word 16  @32-bit variable val3 in memory
6        .section .text
7        .globl _start
8        _start:
9         ldr r1,=val1  @load memory address of val1 into r1
10        ldr r1,[r1]   @load the value val1 into r1
(gdb) list
11        ldr r2,=val2  @load memory address of val2 into r2
12        ldr r2,[r2]   @load value  val2 into r2
13        ldr r3,=val3  @load memory address of val3 into r3
14        ldr r3,[r3]   @load value of val3 in r3
15        add r2,r2,r3  @add r2 and r3, store in r2
16        add r2,#9      @add 9 to r2
17        sub r2,r1     @sub r1 from r2
18
19        svc #0        @Program Termination:wake kernel
20       .end
(gdb) b 19
```


Pictures
[arithmetic2.s]
pi@raspberry...
22:57
pi@raspberrypi: ~
File Edit Tabs Help
19 svc #0 @Program Termination:wake kernel
20 .end
(gdb) b 19
Breakpoint 1 at 0x10098: file arithmetic2.s, line 19.
(gdb) run
Starting program: /home/pi/arithmetic2
Breakpoint 1, _start () at arithmetic2.s:19
19 svc #0 @Program Termination:wake kernel
(gdb) stepi
9 ldr r1,=val1 @load memory address of val1 into r1
(gdb) x/3xw 0x10098
0x10098 <_start+36>: 0xef000000 0x000200a8 0x000200ac
(gdb) info registers
r0 0xfffffffc 4294967292
r1 0x6 6
r2 0x1e 30
r3 0x10 16
r4 0x0 0
r5 0x0 0
r6 0x0 0
r7 0x0 0
r8 0x0 0
r9 0x0 0


Pictures
[arithmetic2.s]
pi@raspberry...
22:58
pi@raspberrypi: ~
File Edit Tabs Help
19 svc #0 @Program Termination:wake kernel
(gdb) stepi
9 ldr r1,=val1 @load memory address of val1 into r1
(gdb) x/3xw 0x10098
0x10098 <_start+36>: 0xef000000 0x000200a8 0x000200ac
(gdb) info registers
r0 0xfffffffc 4294967292
r1 0x6 6
r2 0x1e 30
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 0x7efff3b0 0x7efff3b0
lr 0x0 0
pc 0x1009c 0x1009c <_start+40>
cpsr 0x10 16
fpscr 0x0 0

<u>Parallel Programming Skills Foundation: Arteen Ghafourikia</u>

➔ **Identifying the components of the raspberry PI B+.**
The components of the raspberry PI B+ are 2 USB ports, ethernet controller, ethernet port, CPU/RAM, camera port, 2 power ports, HDMI and the display ports.

➔ **How many cores does the Raspberry Pi's B+ CPU have?**
- The Raspberry Pi B+ CPU has four cores which is also known as a quad core.

➔ **List three main differences between X86 (CISC) and ARM Raspberry PI (RISC). Justify your answer and use your own words.**
- Intel's instruction set is different because it has a larger and more detailed instruction set that allows complex instructions to access the memory. It adds more versatility but also has less registers than ARM.
- Arm has a more simplified instruction set. ARM only uses instructions to operate on registers and uses load/store instructions to access the memory.
- Another notable difference is the difference in syntax and that both processors require a different form of assembly to execute programs.

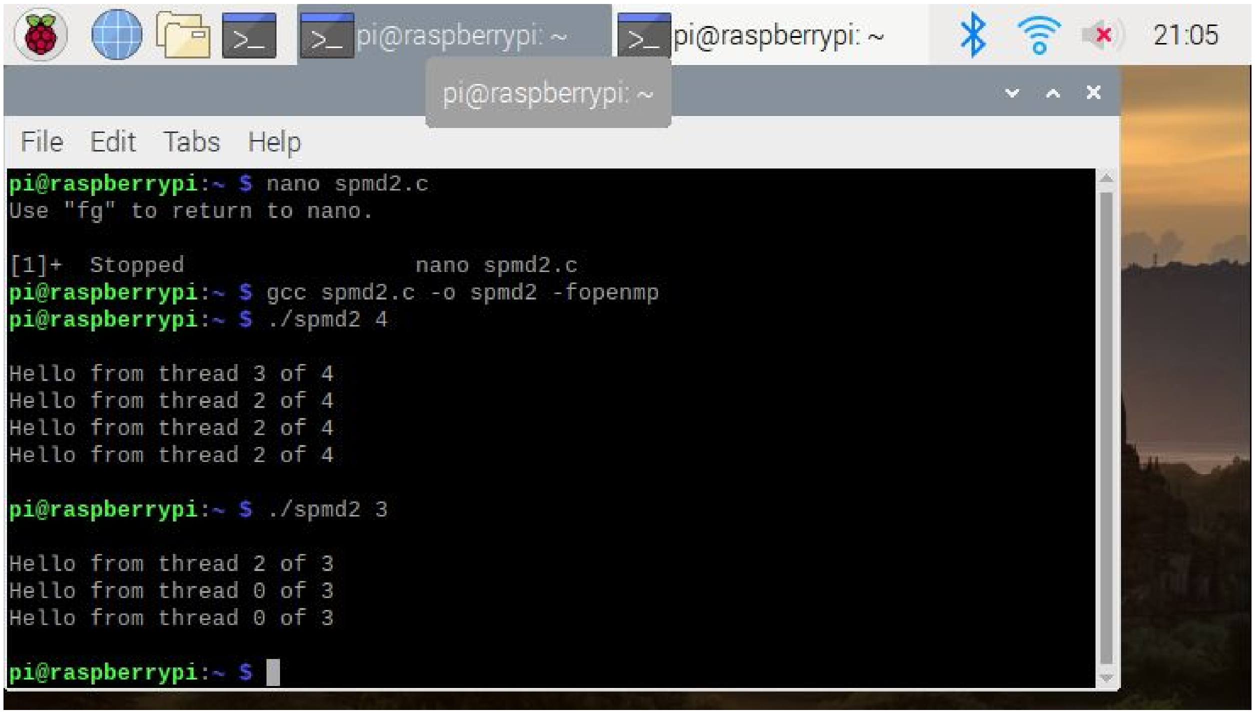
➔ **What is the difference between sequential and parallel computation and identify the practical significance of each?**
- Sequential computation is executed on a single processor and only one instruction can be executed at a time, while parallel computing is broken down to a series of instructions where each part executes simultaneously on different processors.

➔ **Identify the basic form of data and task parallelism in computational problems.**
- In data parallelism you get to perform the same computation with the same input size which makes it much more efficient while in task parallelism the solutions are organized around the functions to be performed rather than around the data.

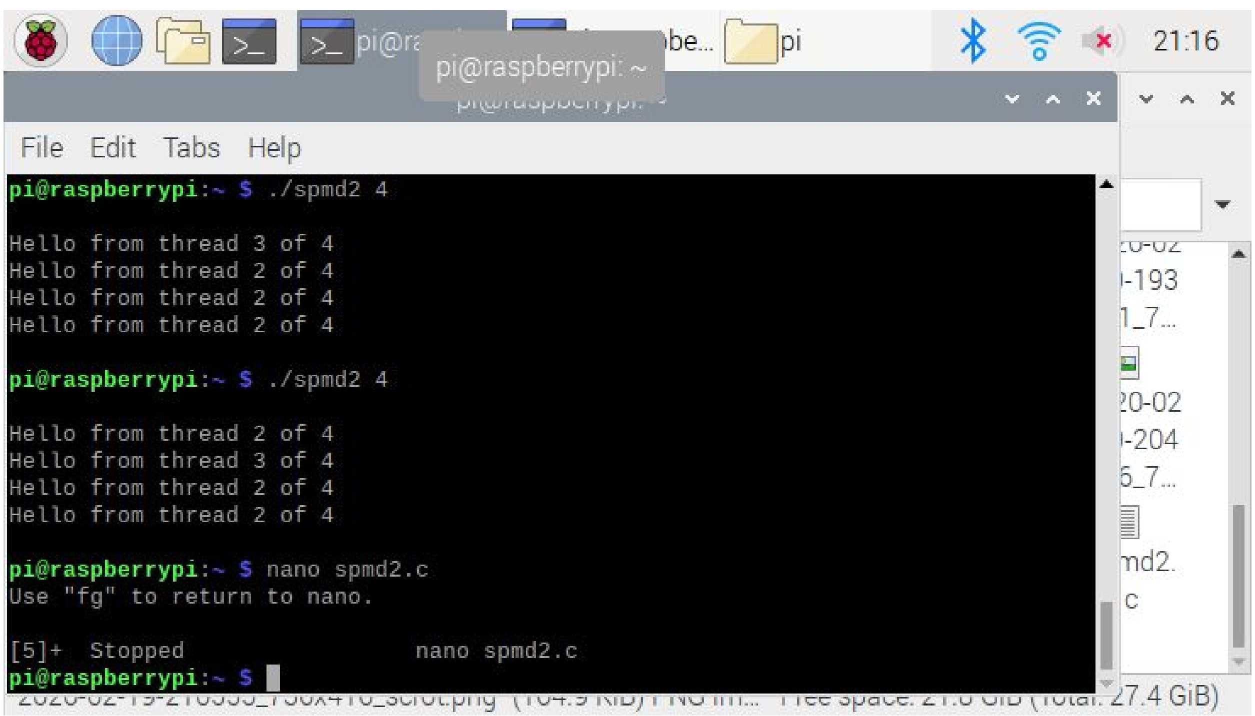
➔ **Explain the differences between processes and threads.**

    - Processes are the abstraction of a running program while Threads are broken up processes that allows single executables to be broken into smaller parts.
- **What is OpenMP and what is OpenMP pragmas?**
    - OpenMP is a library for parallel programming and OpenMP pragmas are the compilers that control how the program works.
- **What applications benefit from multi-core (list four)?**
    - Compilers
    - Scientific applications
    - Web Servers
    - Multimedia applications
- **Why Multicore? (why not single core, list four)**
    - Multicore is faster
    - Single core has head problems
    - Single core has difficult design and verification
    - Single core has speed of light problems

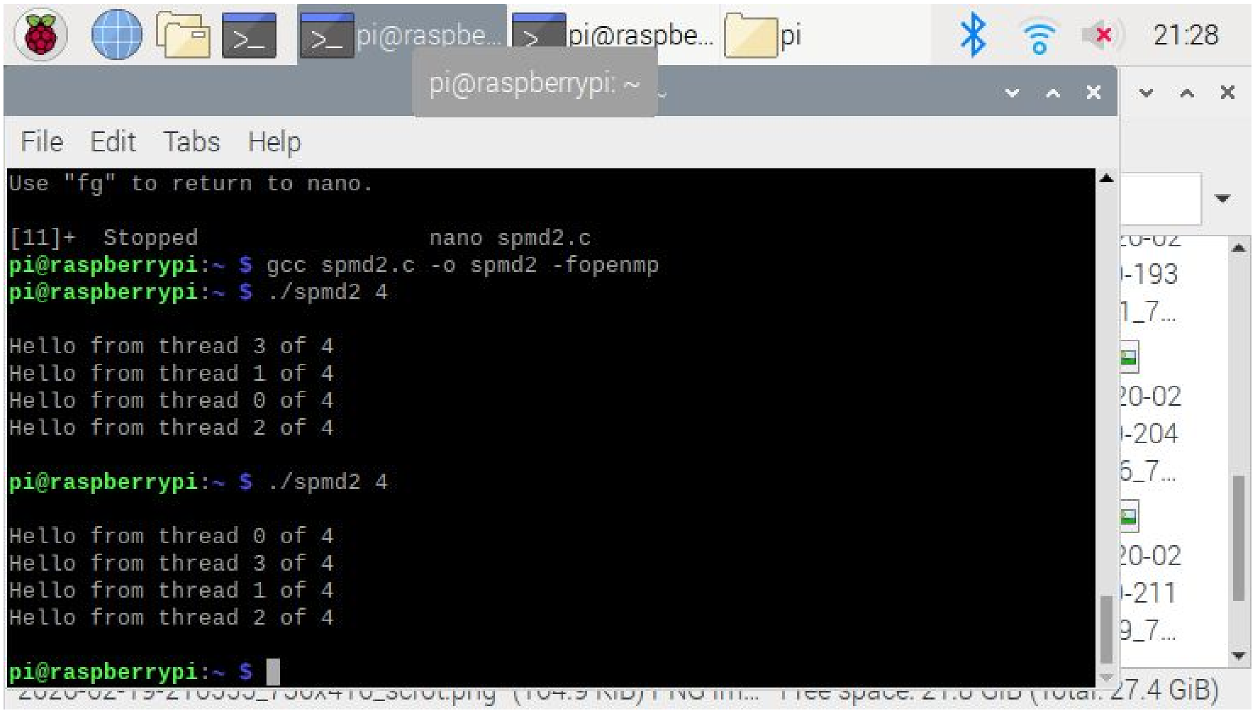
<u>Parallel Programming Basics: Arteen Ghafourikia</u>



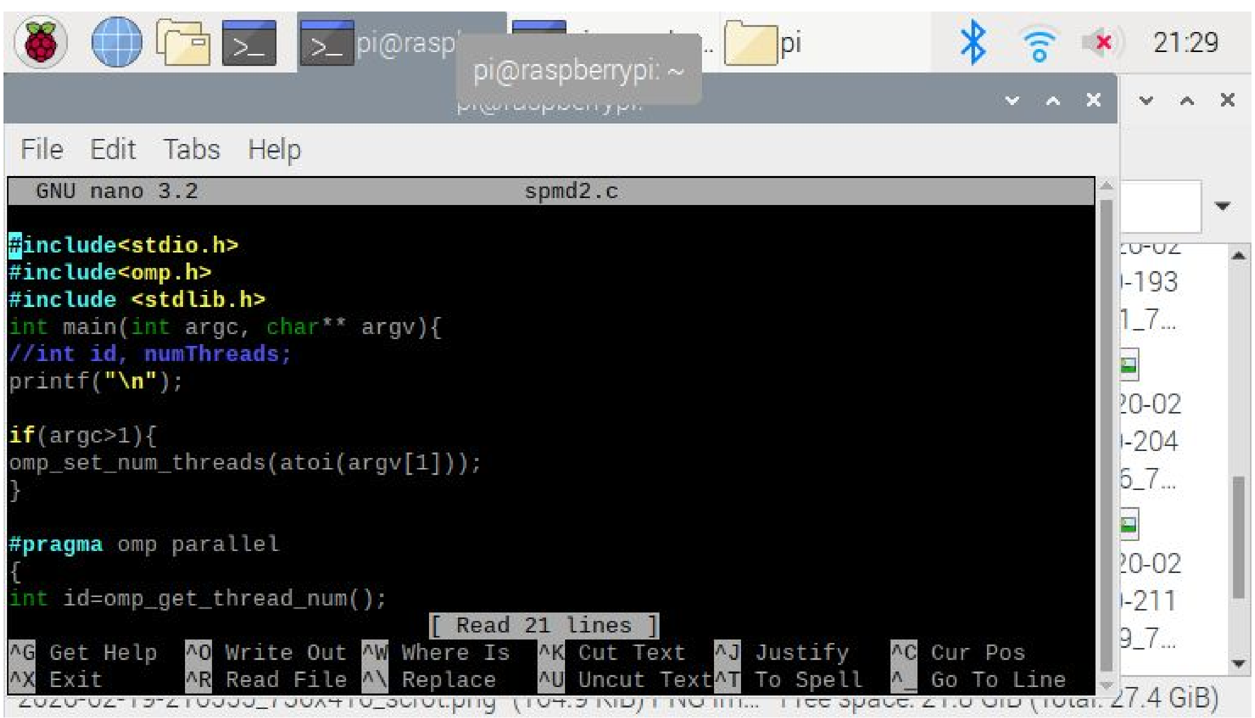
In this screenshot I ran the C program that was given using ./spmd2 4 and realized that the thread ids are all the same. I also alternated the number of threads, but still had threads that had the same id.

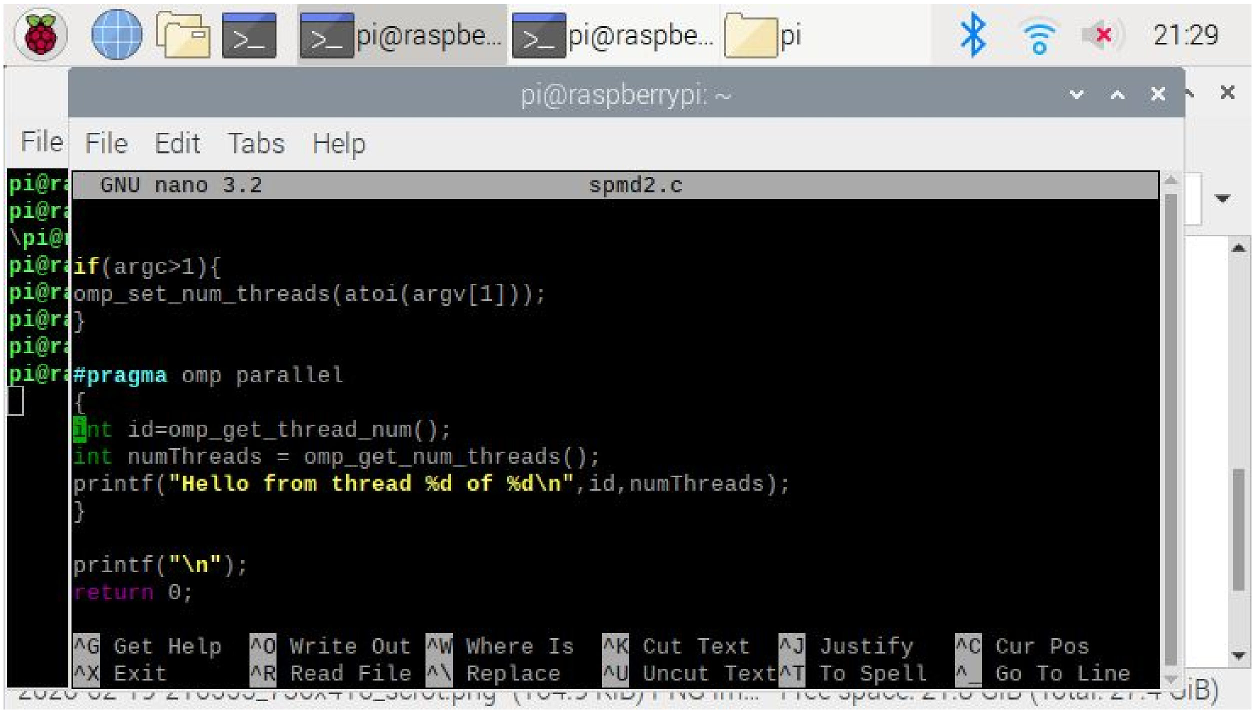


In this screenshot I tried it a couple more times to see if the thread ids would always be the same, and it turned out to be true.



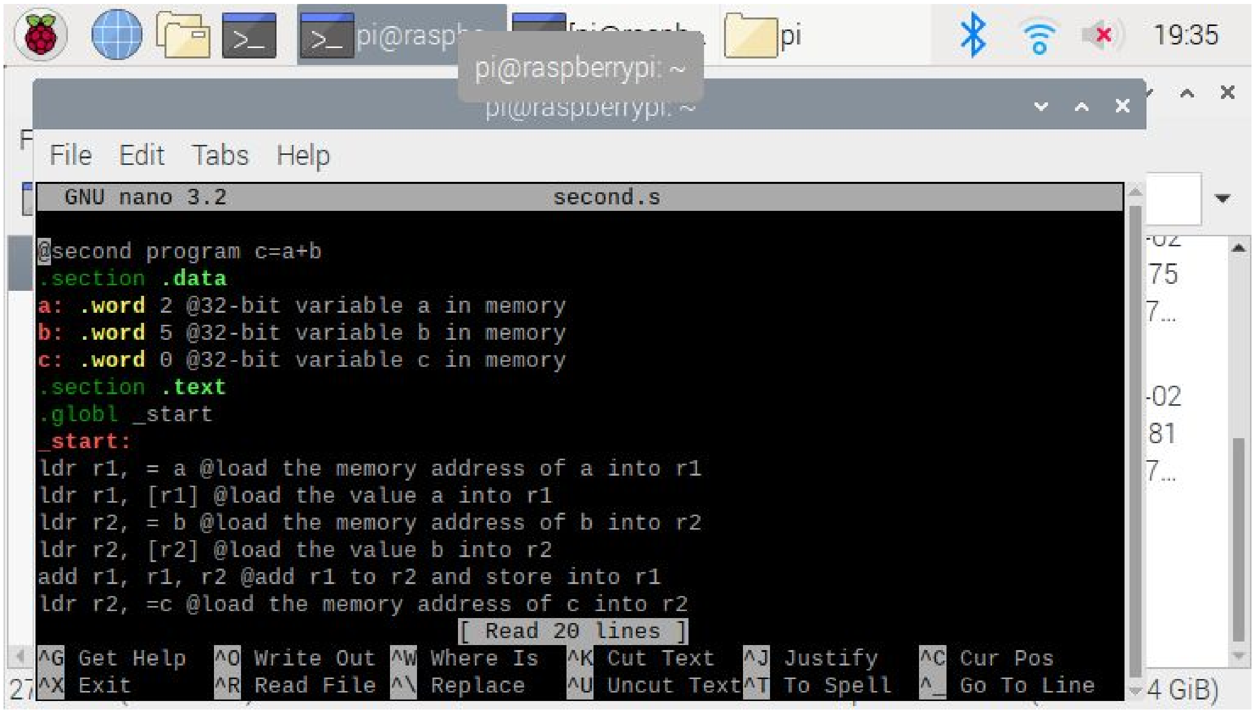
In this screenshot I ran the program after I made the changes to the code, and the thread ids were different. The problem with the code was that it was not initializing a new variable, but reusing the same one. However, after I made the changes, the thread id displayed different values.



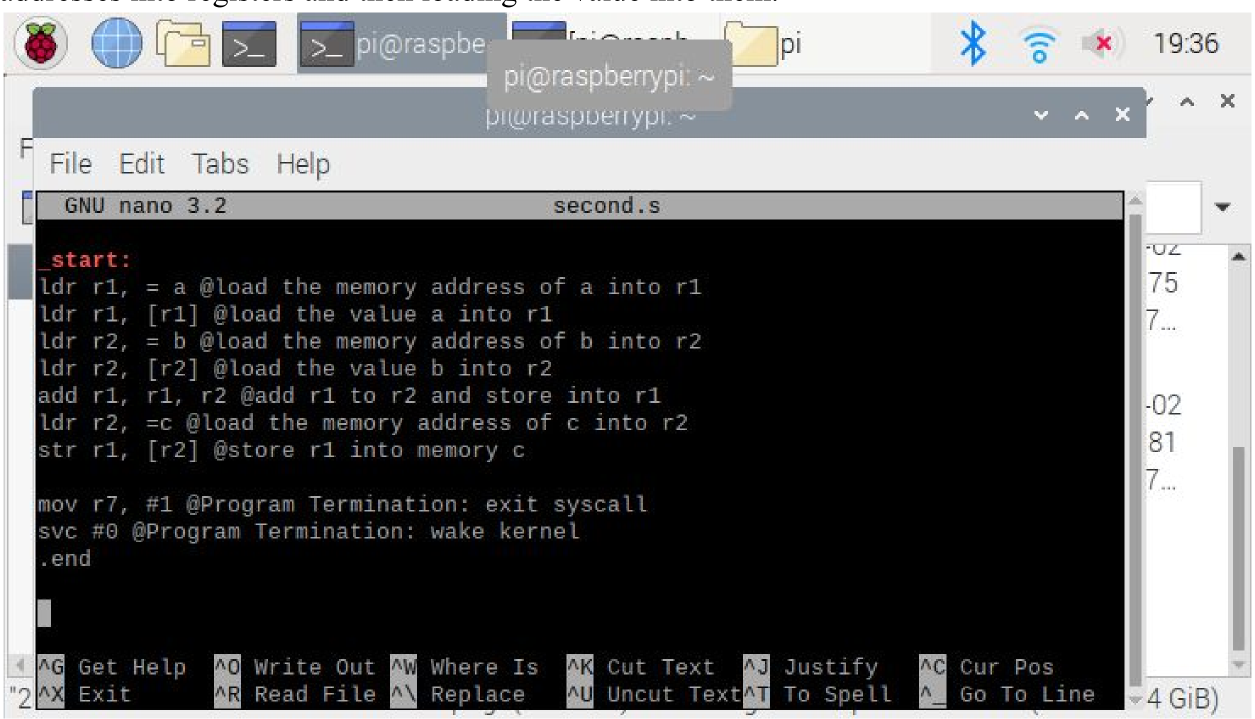


These two screenshots show the correct version of the code. The only difference between the correct version of the code and the old version is that the new version initializes a new variable giving us different thread ids, while the older one uses the same thread id which would basically make it not a thread.

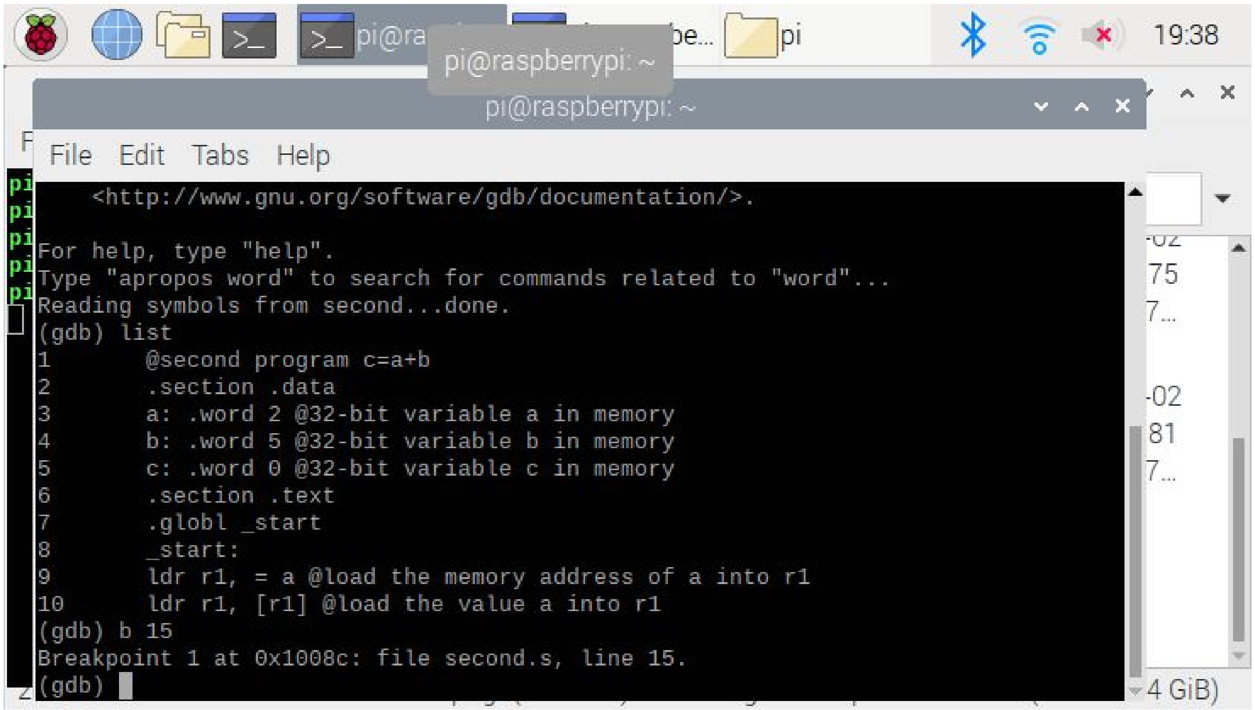
Arm Assembly Programming:Arteen Ghafourikia



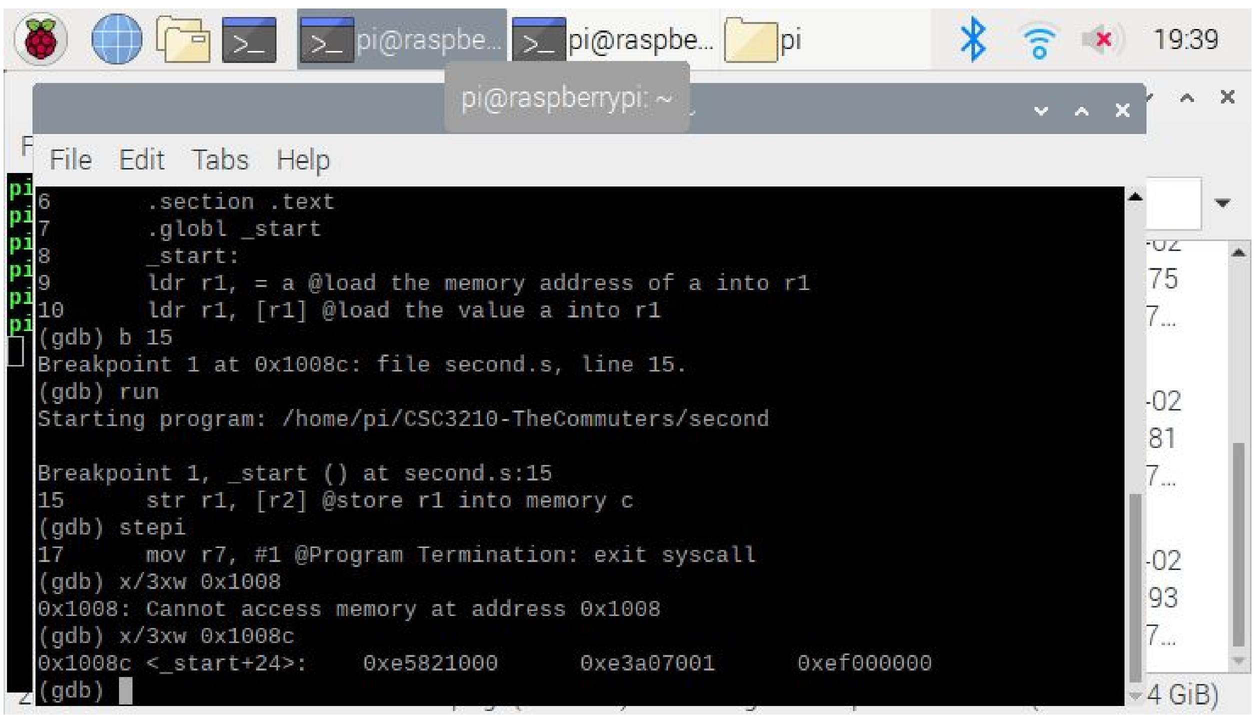
Here is the example code for part 1 that was given. In this code we are loading the memory addresses into registers and then loading the value into them.



Here is the rest of the example code for part 1 that was given.

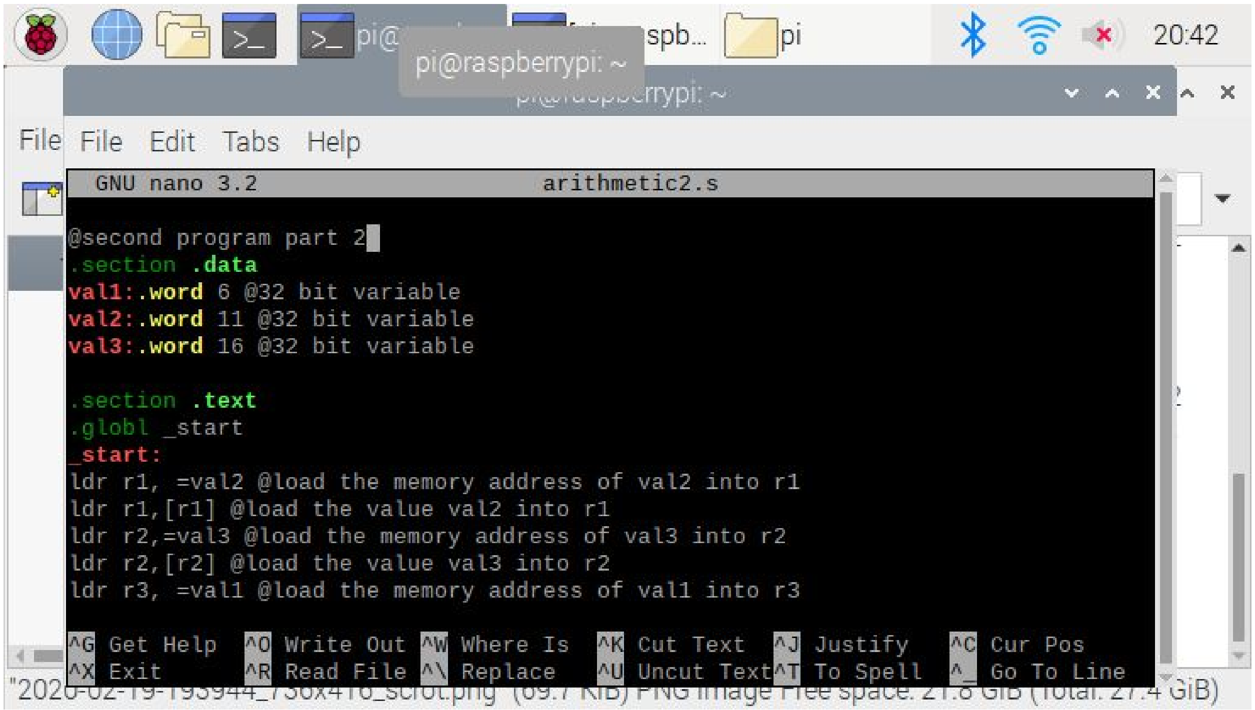


In this screenshot I listed the first 10 lines of code with (gdb) list and then I placed a breakpoint at line 15.

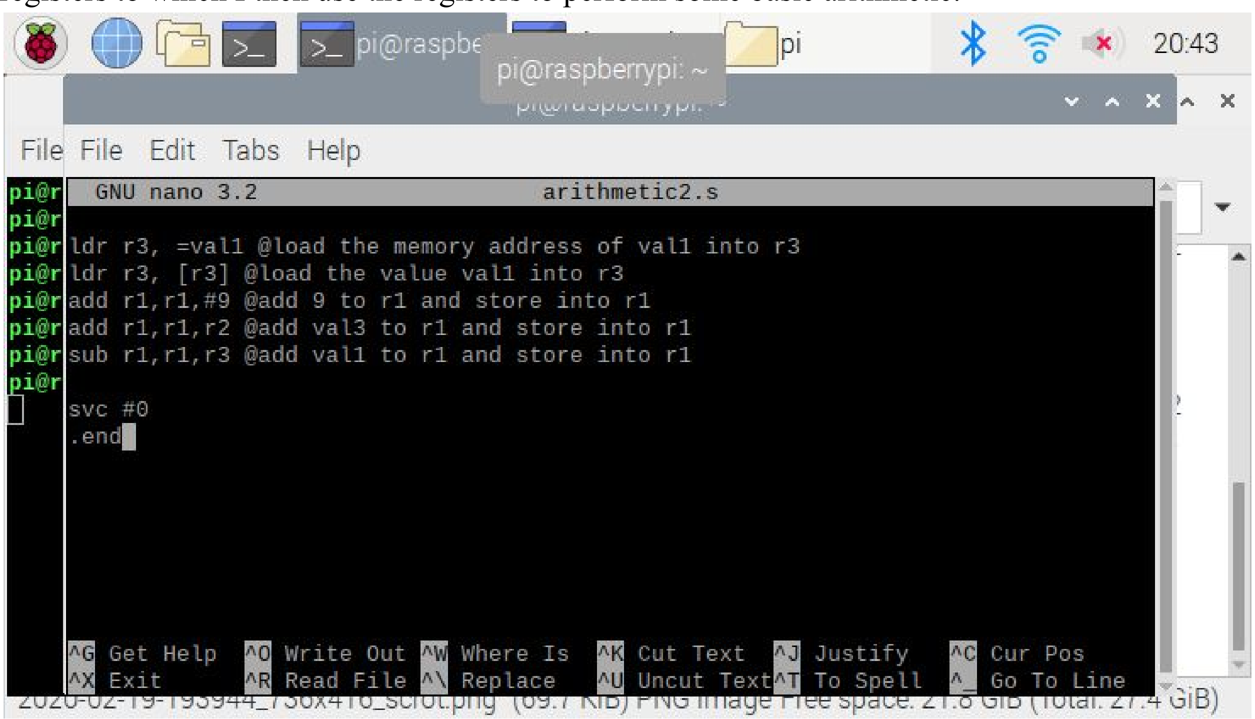


In this screenshot I ran the program and then stepped into the next line and then displayed the memory addresses to which it then displayed three memory addresses on the line of code.

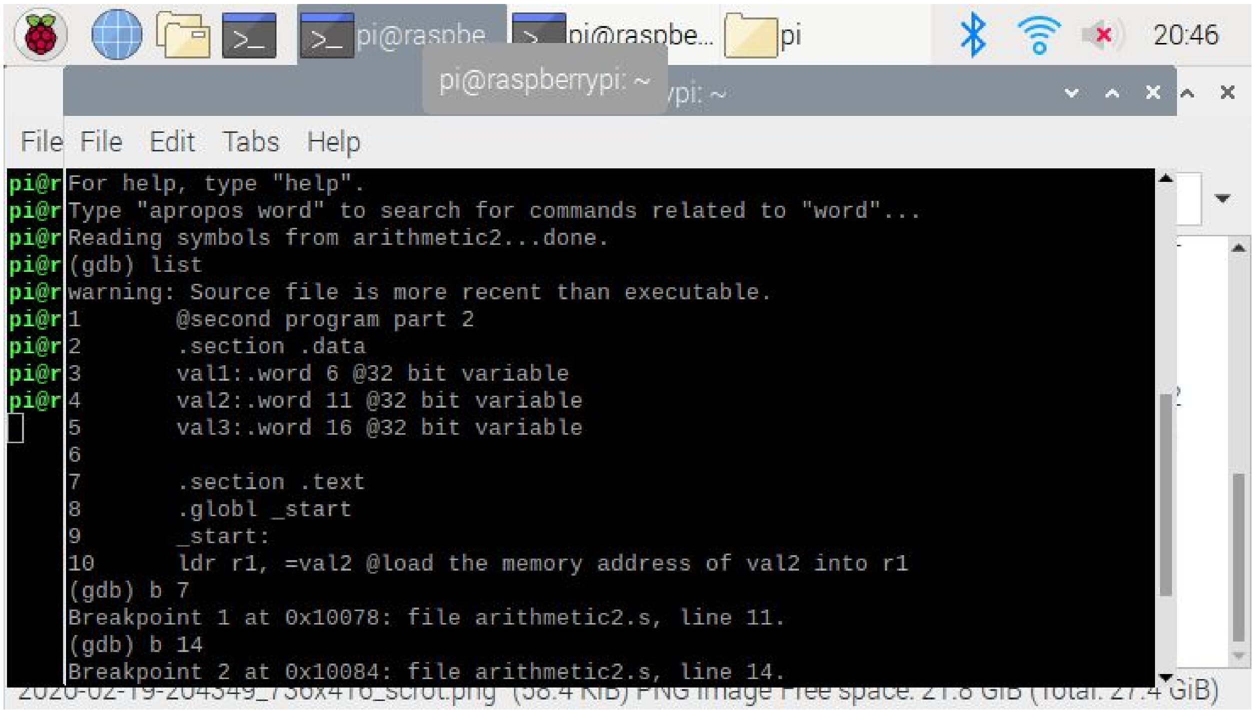
Part 2:



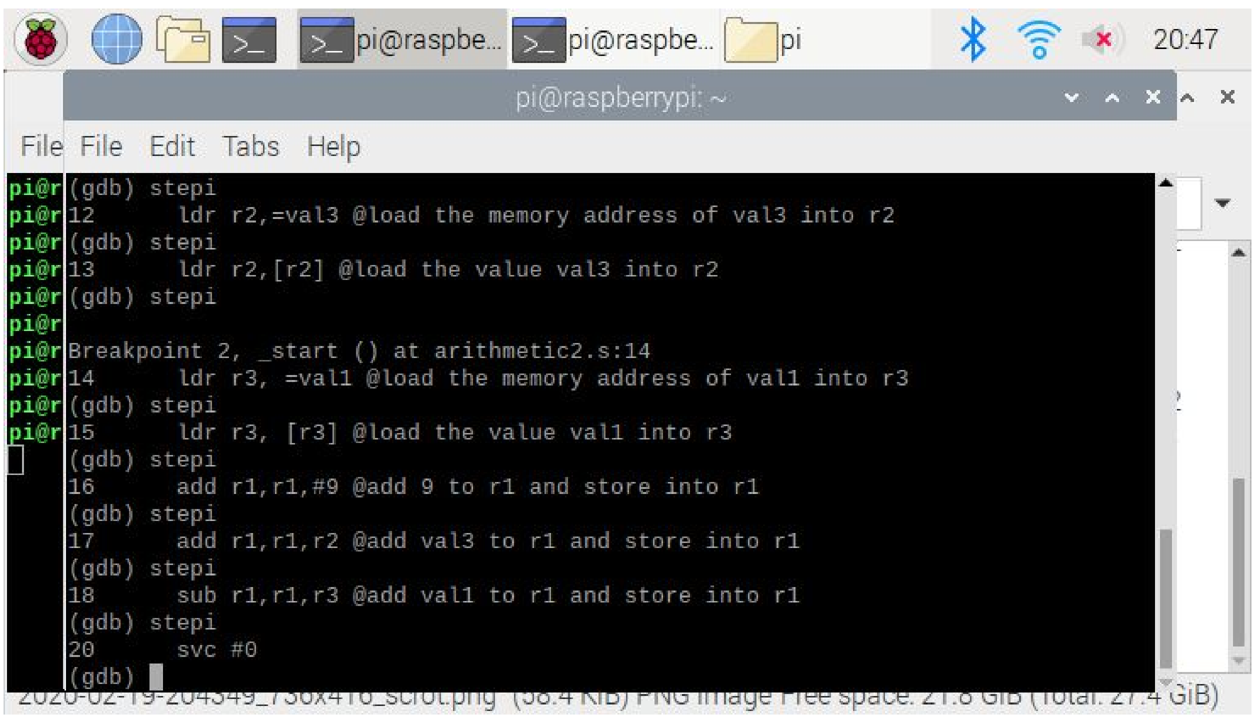
This is the code for the second part. In this program I am loading the memory addresses into registers to which I then use the registers to perform some basic arithmetic.



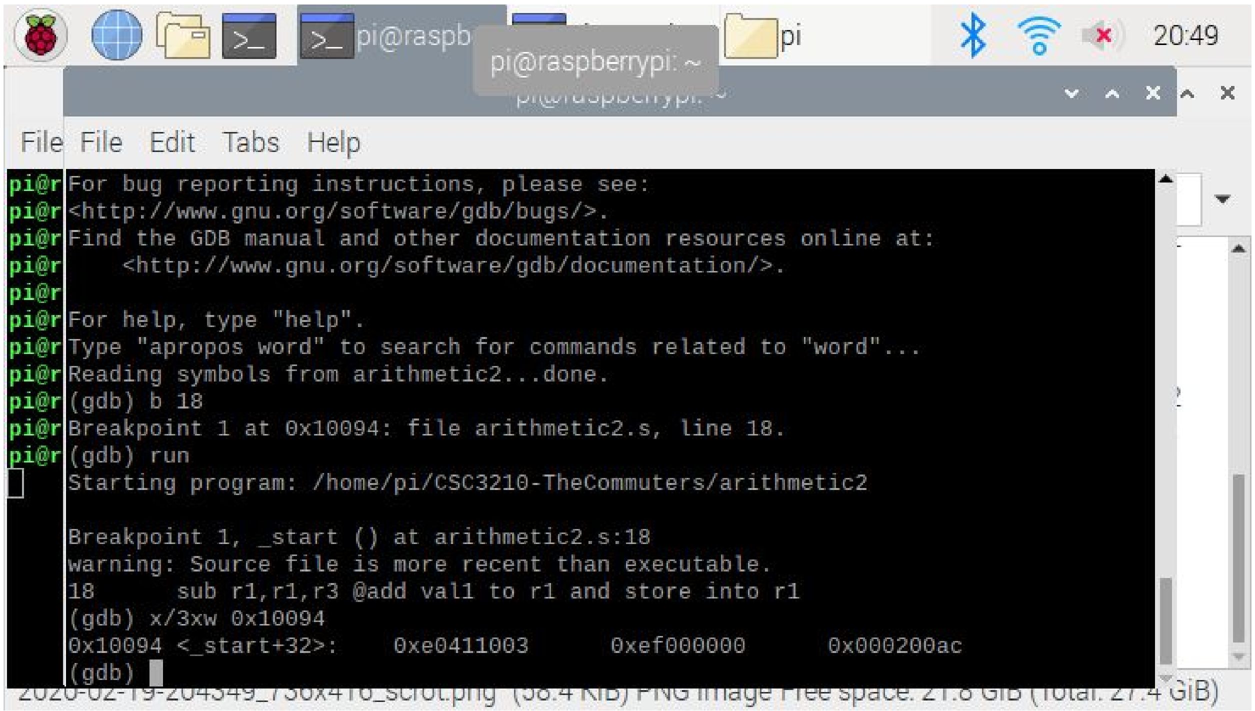
This is the rest of the code for the second part.



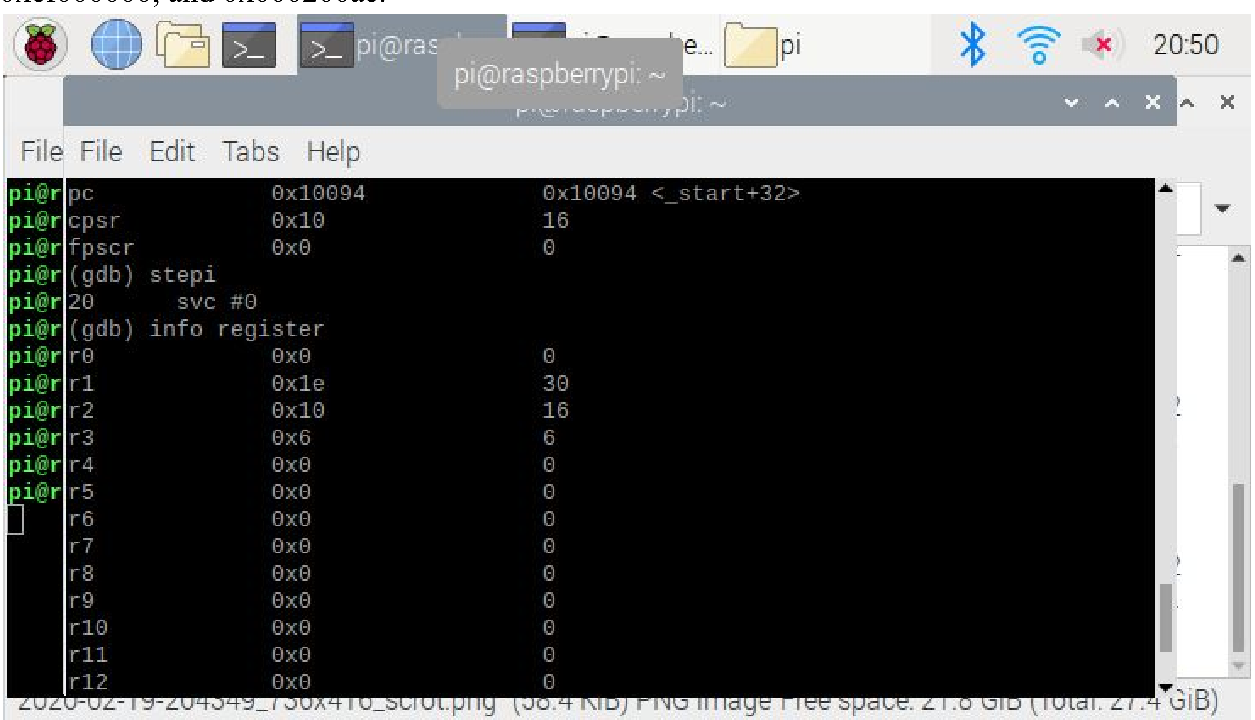
In this screenshot I listed the first 10 lines of my program and then placed a breakpoint at 7 and 17.



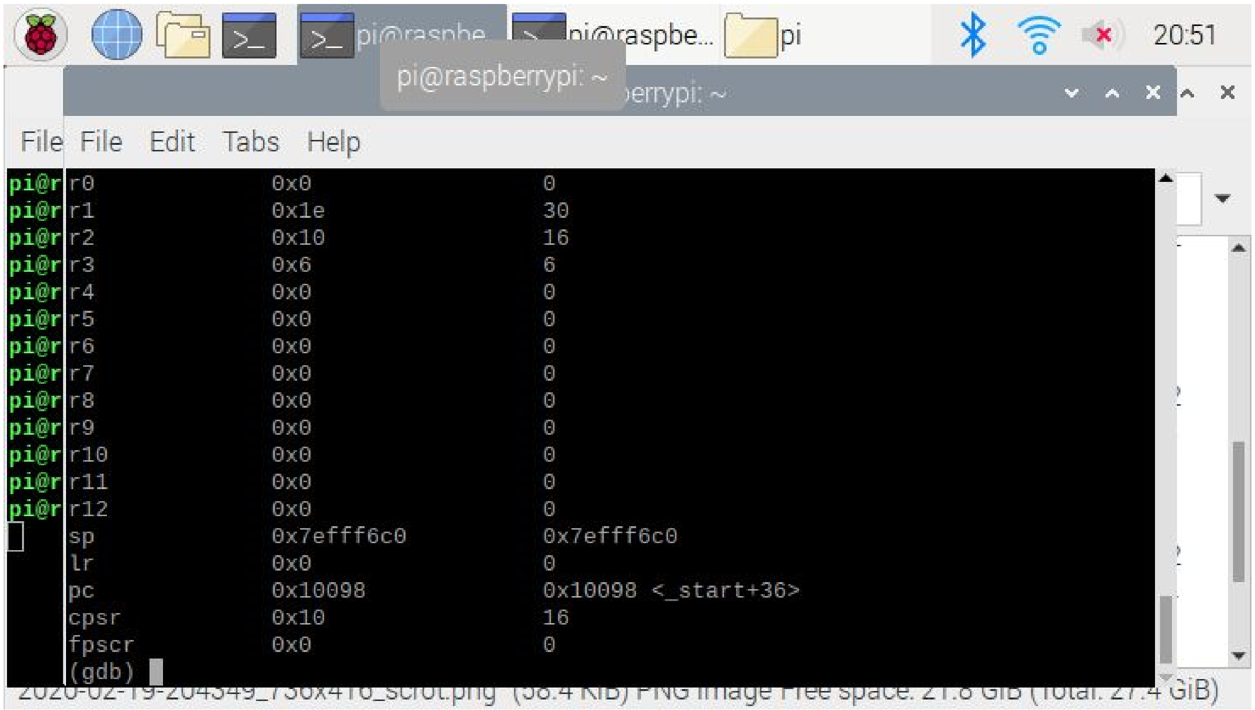
In this screenshot I was stepping into each line of code to see what I would see and it showed each line.



In this screenshot I placed a breakpoint at line 18 and then ran the program. I then displayed the registers to see what I would get and it displayed 3 memory addresses which are 0xe0411003, 0xef000000, and 0x000200ac.



In this screenshot I displayed the registers using (gdb) info register and as you can see the result in register 1 is 30 which would be the current value if you compute the given arithmetic.



This is a screenshot of the rest of the registers.

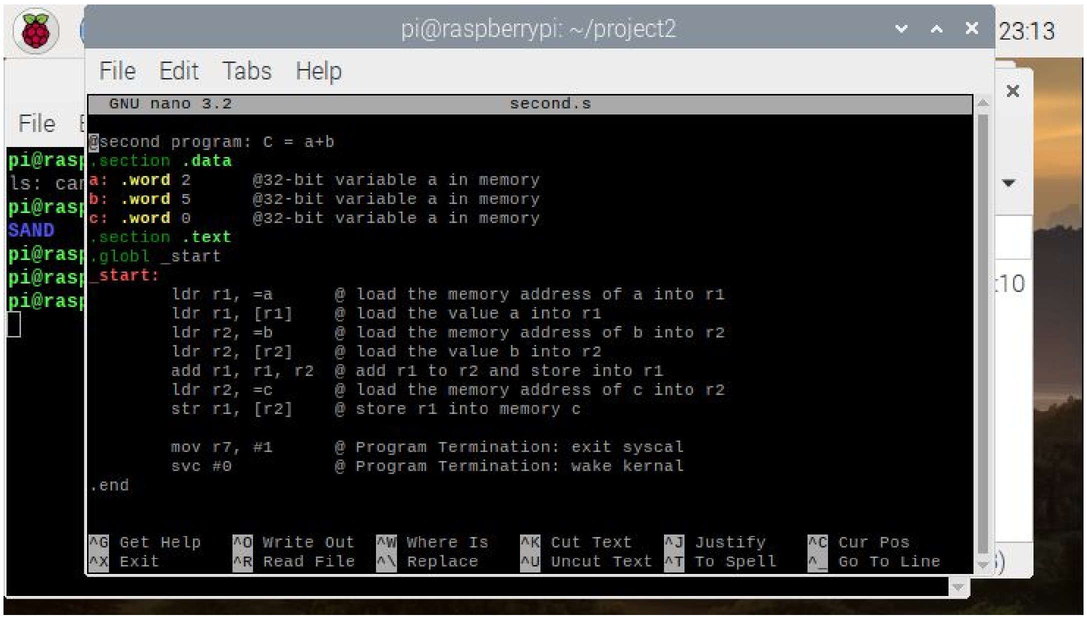
<u>Parallel Programming Skills Foundation: Joan Galicia</u>

- Identifying the components of the raspberry PI B+.
  - The Raspberry Pi's components are a single board computer, quad -core multicore CPU, 1 GB RAM, 2 Usb ports, ethernet/ controller, Power port, Camera, HDMi, Power, and Display.
- How many cores does the Raspberry Pi's B+ CPU have?
  - .The RSP B+ has 4 cores in its CPU as it is denoted by having quad-core
- List three main differences between X86 (CISC) and ARM Raspberry PI (RISC). Justify your answer and use your own words.
  - The major difference between x86 and ARM is that they have different languages. This means that if an instruction set was given and made to the x86, it could not be used for the ARM.
  - Another key difference is their difference in processing power. The x86 was designed to handle larger and complex instructions while ARM was designed to be fast. Arm's memory is based on loading and storing as its instruction sets can only be used in registers.
  - X86 is focused on efficiency while ARM has its focus on fast execution for less clock cycles per instruction

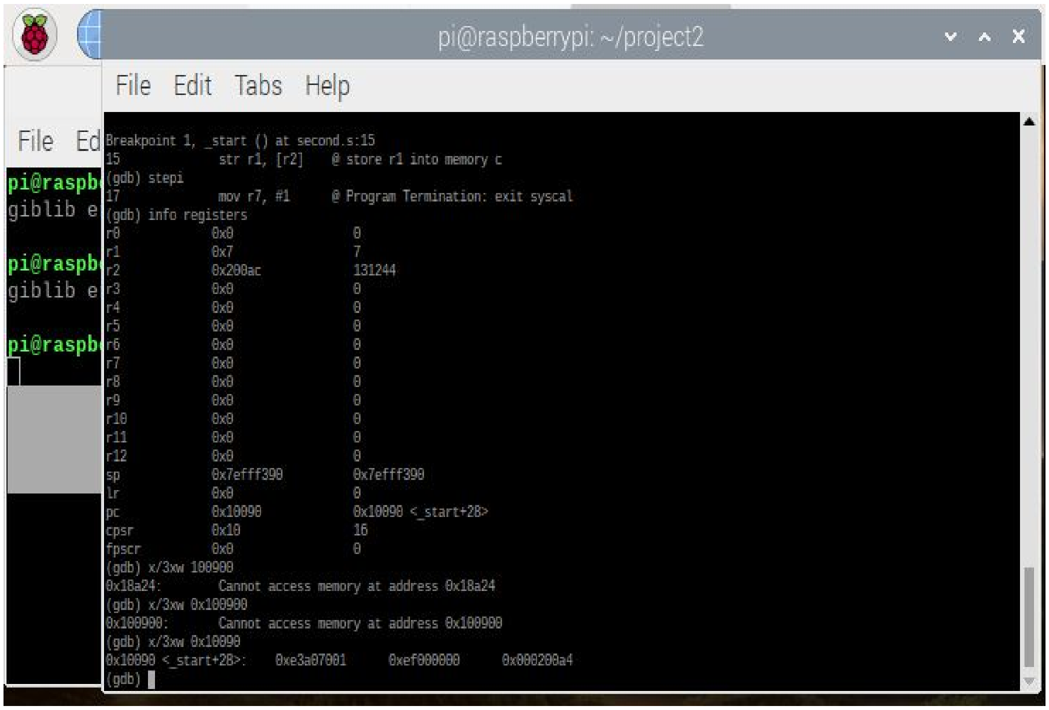
- What is the difference between sequential and parallel computation and identify the practical significance of each?
    - In sequential computation a task is runned through a processor and completed one by one. Its Purpose is to run similar tasks that are completed at similar times. The set back is that it takes more time to complete the tasks because only one processor is being utilized.
    - Parallel computation is a way for multiple tasks to be completed. It is an efficient method because it uses multiple processors that break down tasks into a series of instructions and they run at the same time.
- Identify the basic form of data and task parallelism in computational problems.
    - Data Parallelism is where the same type of data is split among the processors and each of the processors are doing the same calculations with their own piece of data. An example of this, where you are given an image and you want to know how many pixels there are. This would mean that the process needed to count the number of pixels would be the same and you would just need to count every individual pixel.
    - Task Parallelism is where multiple tasks are needed to be executed and they have different functions. This parallelism is based on the task itself and not solely on the data within the task. An example of this, is where you have an array with some amount of integers and you would like to find the minimum, maximum, and average of this array. In this case we are looking at the same data but to get these results you would need different processors to look at the data and compute it.
- Explain the differences between processes and threads.
    - Both processes and threads are part of the execution of a program that run through the memory, but the differences will be given below:
    - Processes run in their own memory space and are given everything they need to execute a program even having one thread of execution. The way threads are different is that it's an entity within a process which is scheduled to be executed, but it has to be in the same memory space.
- What is OpenMP and what is OpenMP pragmas?
    - OpenMP is a library that uses a thread pool pattern of simultaneous execution controls that supports shared memory and data multiprocessing.
    - OpenMP pragmas is an implicit multithreading compiler where the library will control thread creation. It is a low level compiler that helps programmers become less error prone.
- What applications benefit from multi-core (list four)?
    - Compliers
    - Scientific applications
    - Web servers
    - Any applications with thread level parallelism
- Why Multicore? (why not single core, list four)

- Multiple single process executions
- Increased inputs of system
- Faster execution
- Power consumption problem is reduced allowing for an increased frequency scaling

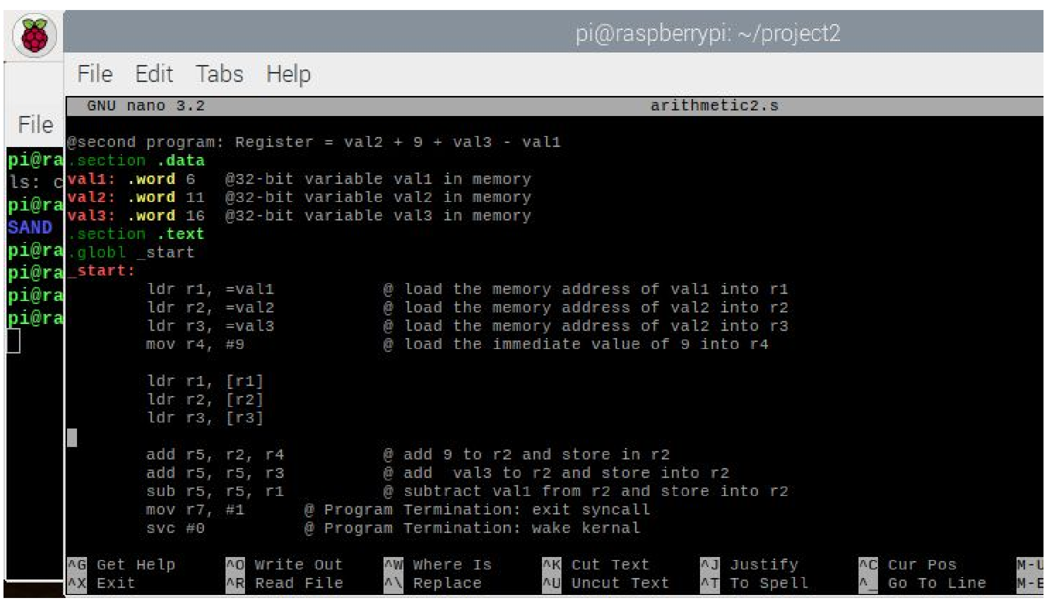
Parallel Programming Basics: Joan Galicia



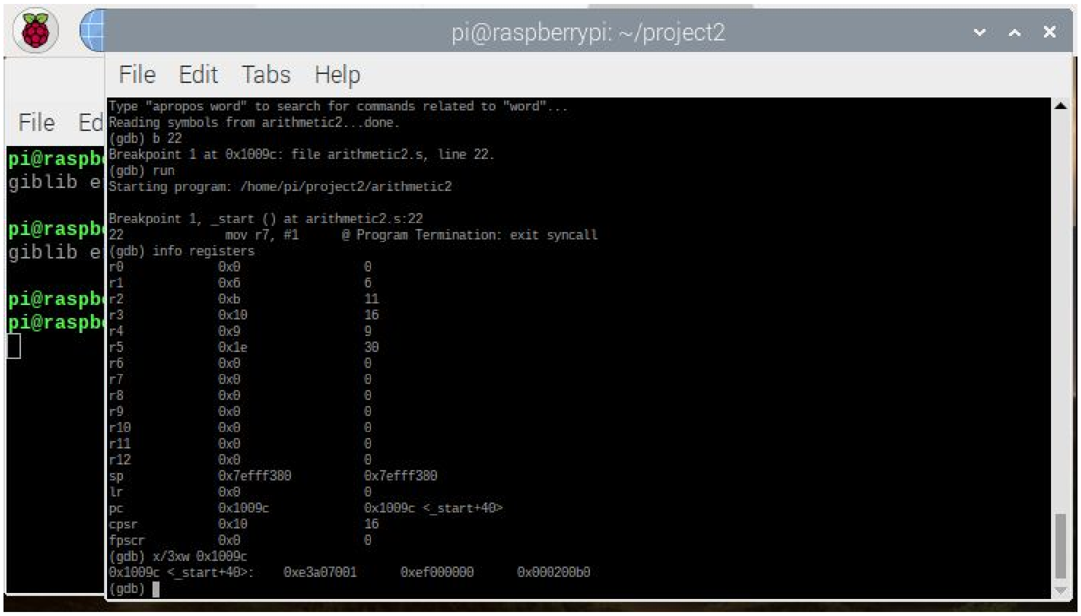
In this screenshot is the code written for second.s in the next screen shot is where the code is debugged. The only difference in this program is the way variables are used and called for. This language uses first assigns the memory address to the register and then uses load instead of move to assign the variables into a register.



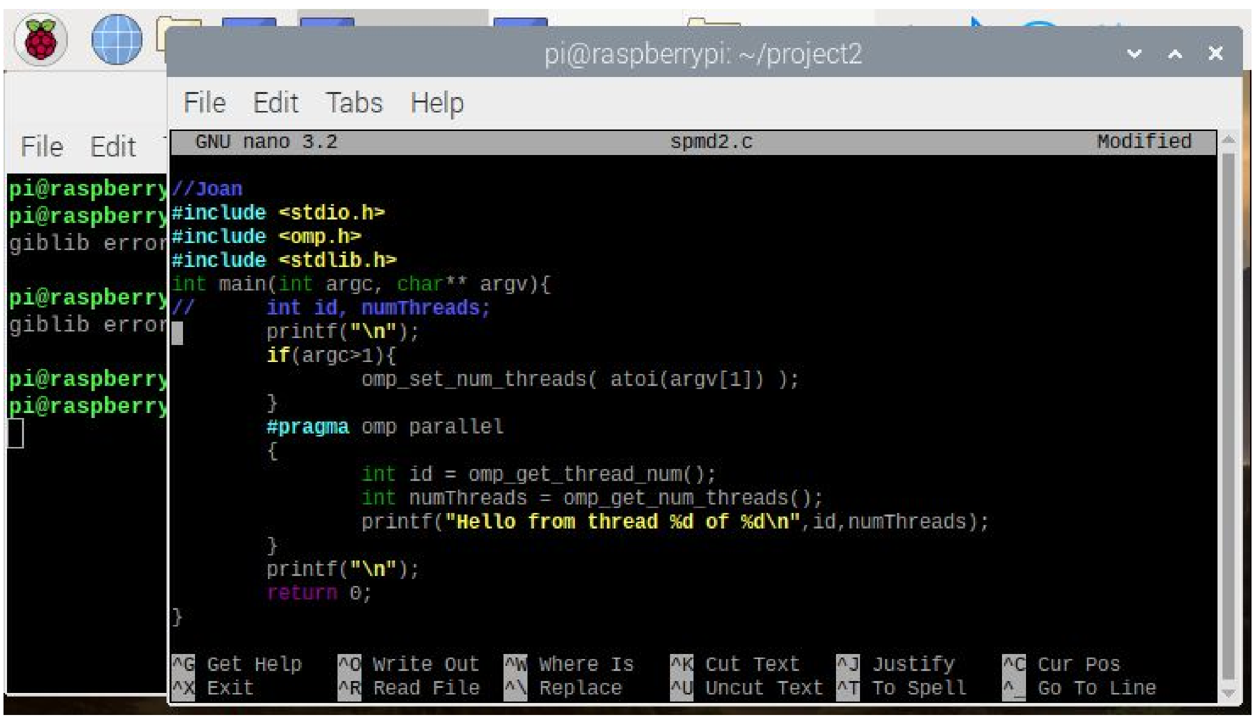
At breakpoint 15 the memory address resulted in 0x01008 and once "stepi" was commanded it then proceeded to the next line where the code is. This screenshot is set up at line 17 and so the memory registers gave this address 0x10090. After accessing this address it gave 3 more addresses that when trying to access; it did not execute.



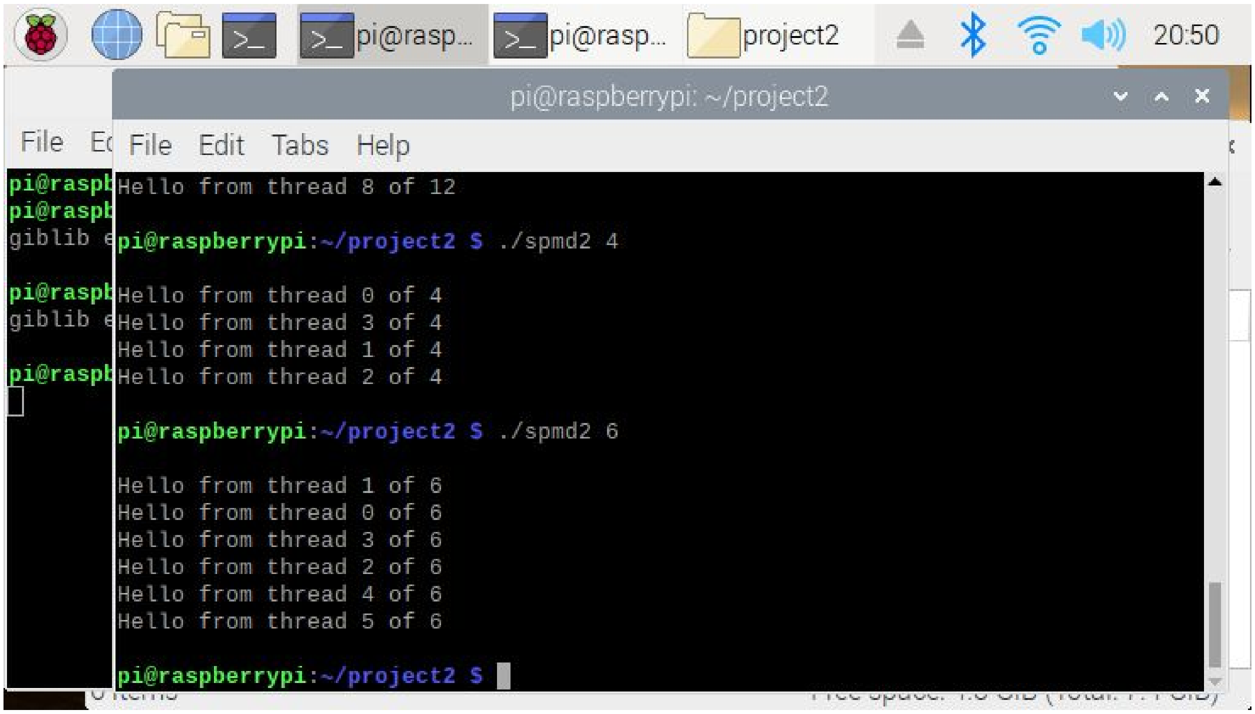
This program is arithmetic2 where using the things I learned in the second program I loaded the values in the variables to the registers. This program was asking to add three values and subtract one value.



The image above is arithmetic2 debugged. The values given and are shown in registers r1, r2, r3, and r4. The result (11+9+16-6) is located in register r5 which is 30 in decimal and in Hexadecimal the answer is 1E. I also accessed the memory access (seen in register pc) where it would give me three more addresses.



This screenshot is displaying the code for a new program I have started working on called spmd2.c. This program is different because it is written in the C Language, and is used to show how parallel programming works. The errors that I came across from this program was that



In this last screen show the output of the spmd2.c program is displayed. Before I was able to obtain this output, the program spmd2.c had to be compiled. I compiled the program using the GNU Compiler Collection where it created an executable program that the computer can use.

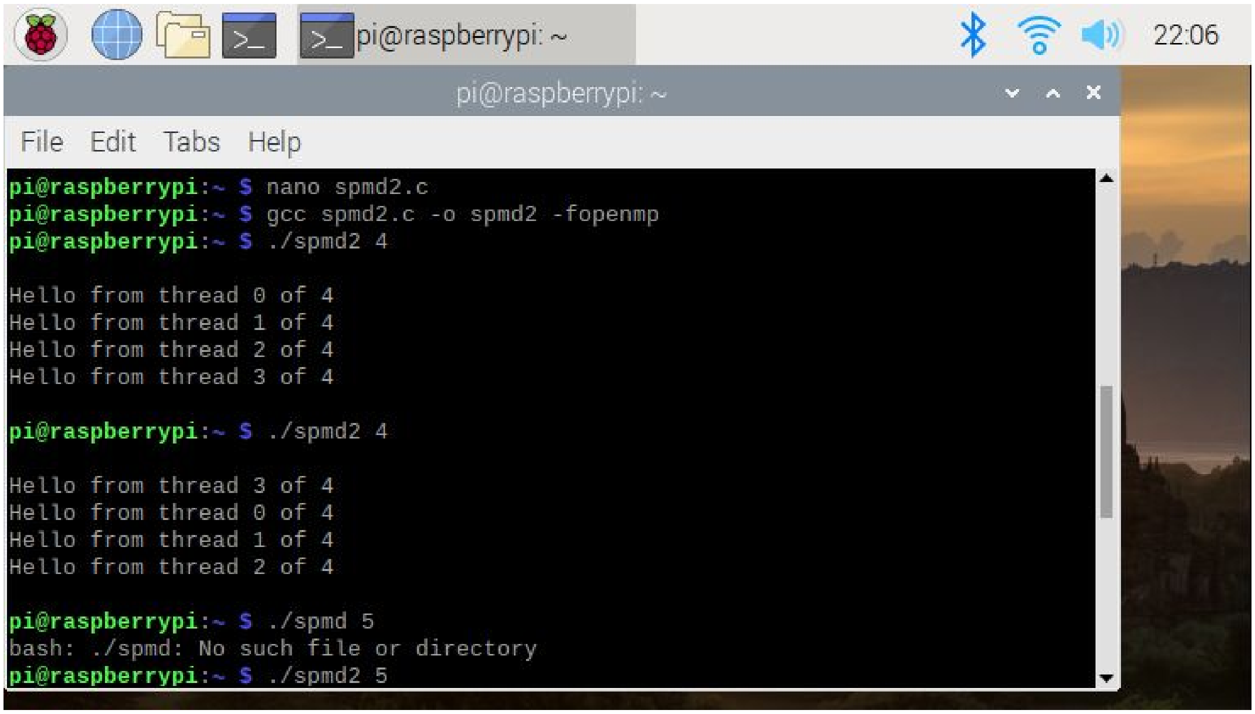
Parallel Programming Skills Foundation: Andre Nguyenphuc

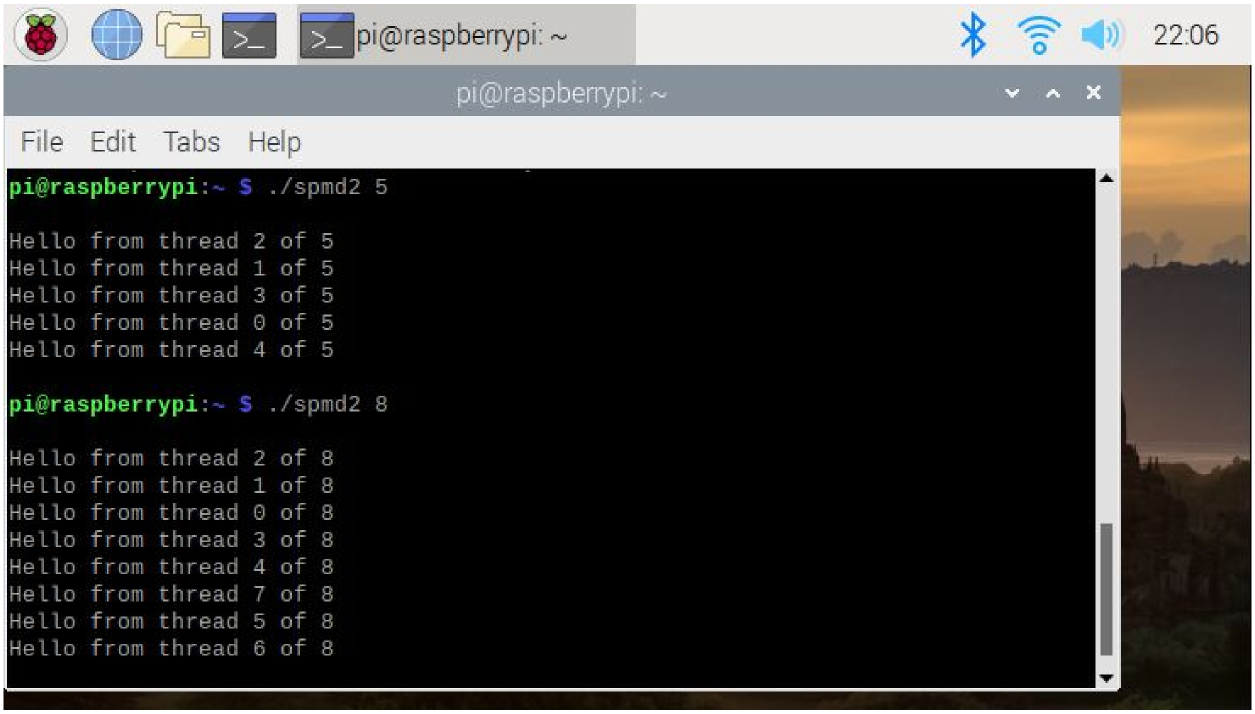
- Identifying the components on the Raspberry PI B+
  - Display port, Power port, CPU/RAM, HDMI port, Camera port, Second power port, Ethernet port, Ethernet controller, 2 USB ports
- How many cores does the Raspberry Pi's B+ CPU have
  - Quad-Core Multicore (4)
- List three main differences between the X86 (CISC) and ARM Raspberry PI (RISC). Justify your answer and use your own words
  - One main difference between X86 and ARM is the instruction set. X86 has a larger and more feature-rich instruction set which allows it to have more operations and addressing modes. ARM has a more simplified instruction set which allows it to have more general purpose registers than X86.
  - ARM is also able to use instructions that operate only on registers and uses a Load/Store memory model for memory access. X86 is able to use different and more complex instructions to access memory.

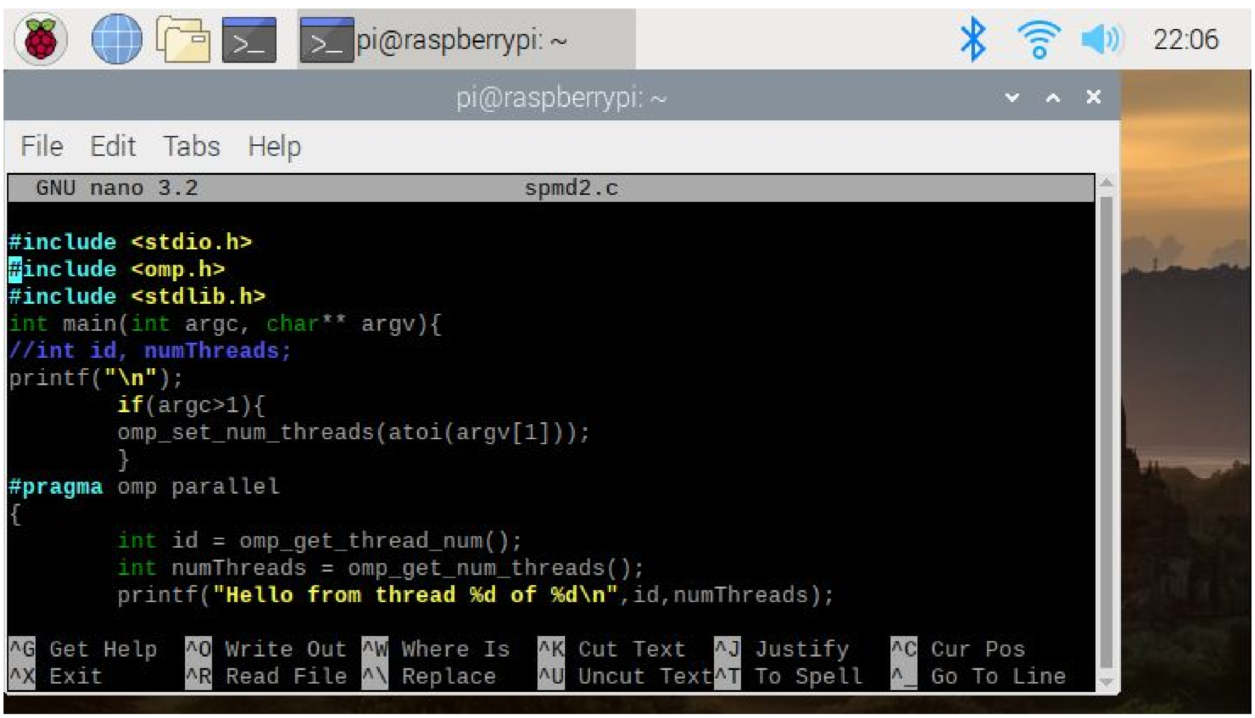
- Another difference between X86 and ARM is that X86 uses the little-endian format while ARM uses BI-endian and includes a setting that allows for switchable endianness.

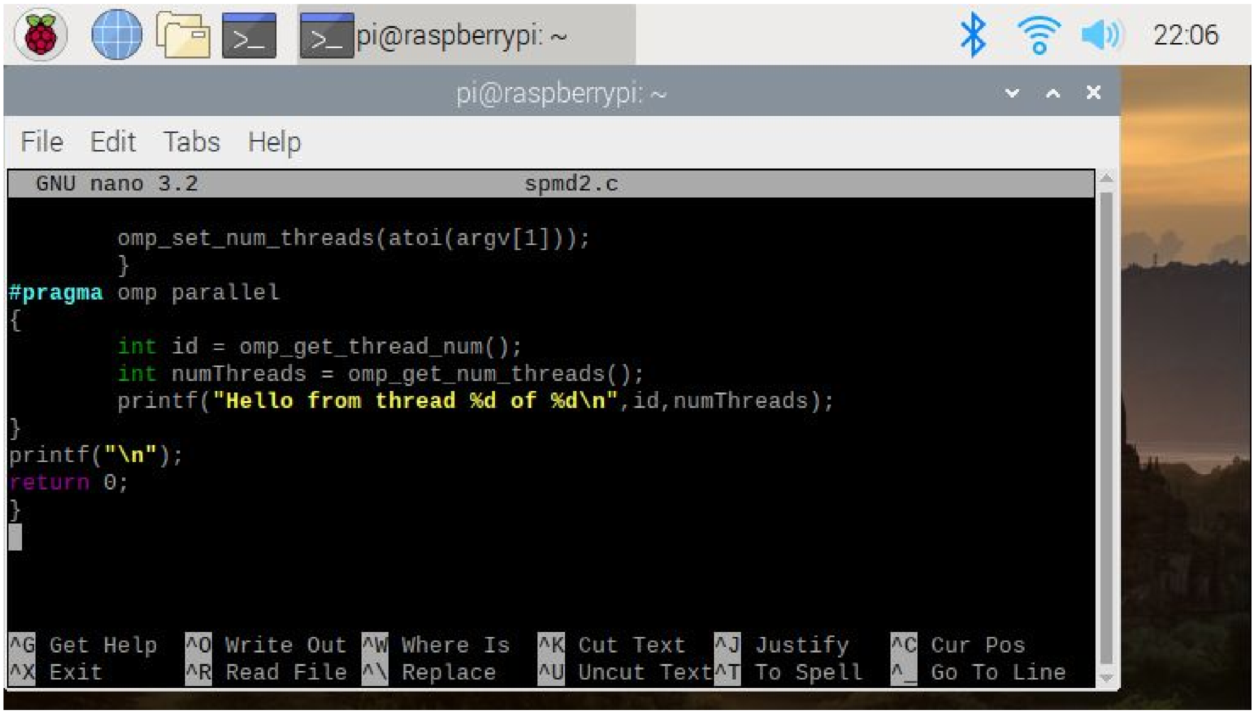
- What is the difference between sequential and parallel computation and identify the practical significance of each?
  - Sequential computation is done on a single processor and breaks down a problem into a series of instructions and each instruction can only be executed one at a time
  - Parallel computation is done on multiple processors and breaks down a problem into seperate parts that can be solved concurrently, and each part can then be further broken down to a series of instructions
- Identify the basic form of data and task parallelism in computational problems
  - Data parallelism
- Explain the differences between processes and threads
  - A process is the abstraction of a running program while a thread is a lightweight process that allows a single executable/process to be decomposed to smaller, independent parts. A difference includes processes do not share memory with each other while threads all share a common memory of the process they belong to.
- What is OpenMP and what is OpenMP pragmas?
  - OpenMP is
  - OpenMP pragmas are compiler directives that enable the compiler to generate threaded code
- What applications benefit from multi-core (list four)?
  - Database servers
  - Web servers
  - Compilers
  - Multimedia applications
- Why Multicore? (why not single core,list four)
  - Many new applications are multithreaded so they need multi-core processors to run
  - Single-core processors’ clocks frequencies are difficult to make even higher
  - Multi-core processors are faster than single-core due to parts of problems being solved concurrently
  -

Parallel Programming Basics: Andre Nguyenphuc

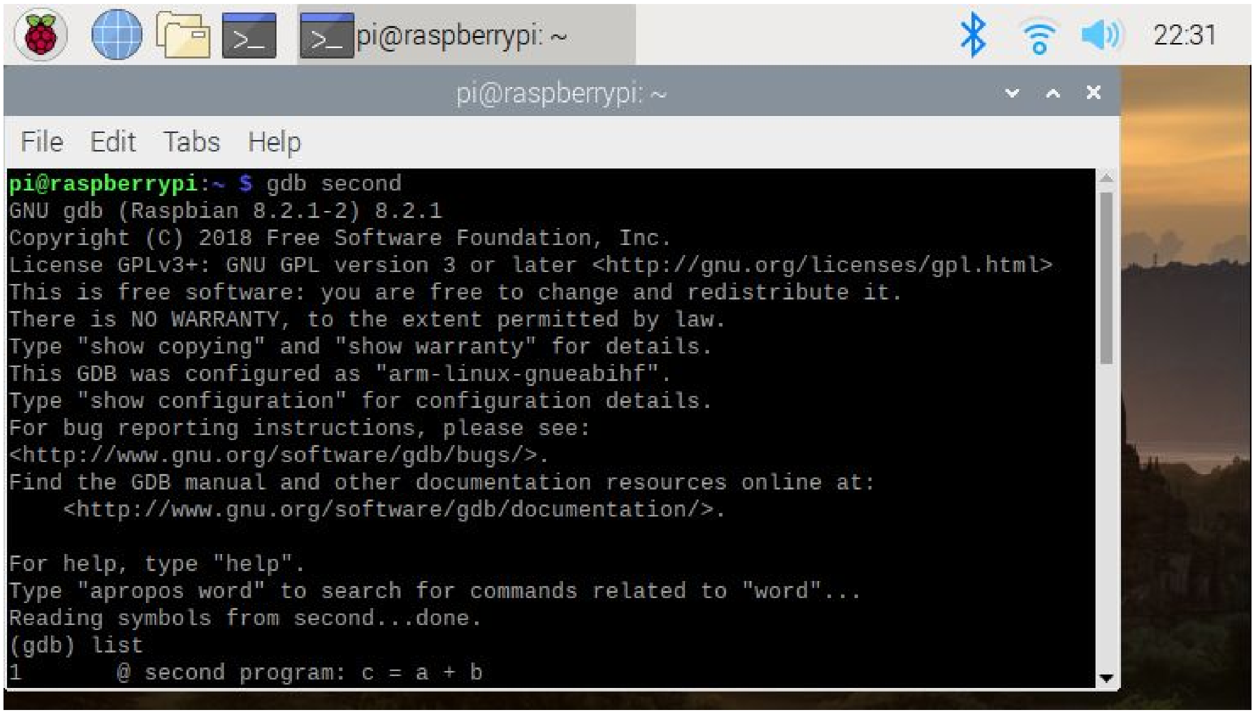


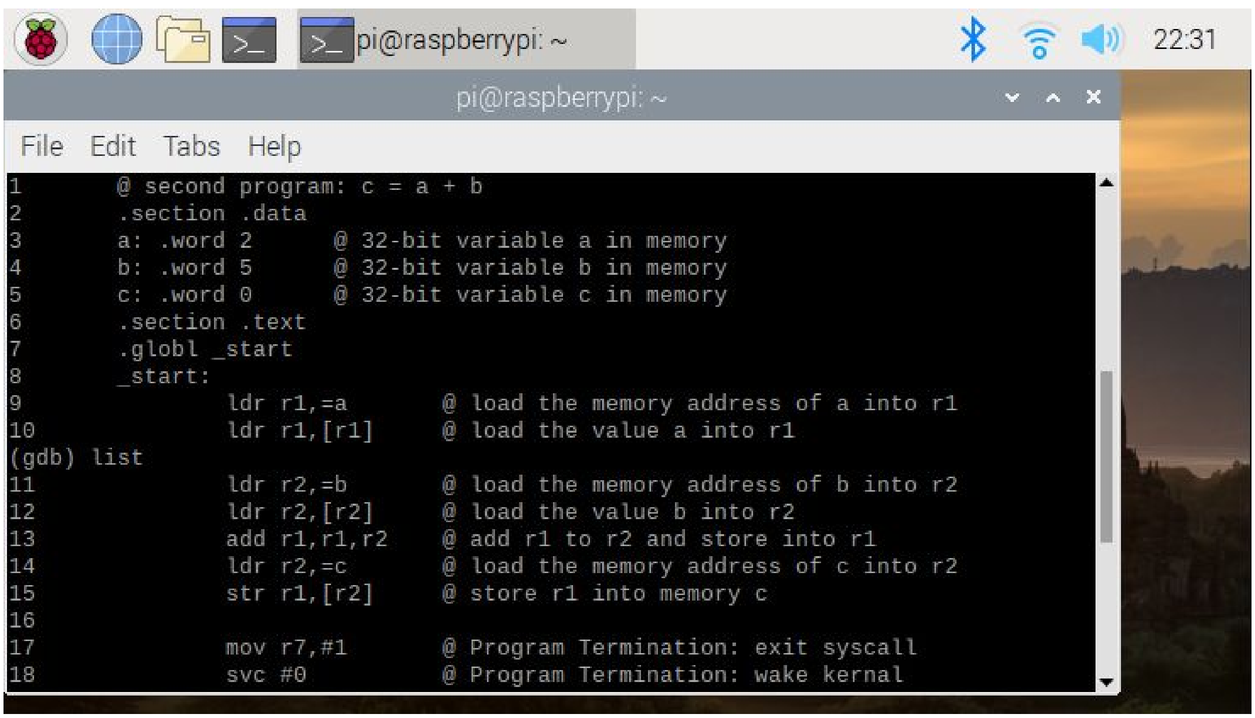

pi@raspberrypi: ~
22:06
pi@raspberrypi: ~
File Edit Tabs Help
pi@raspberrypi:~ $ ./spmd2 5
Hello from thread 2 of 5
Hello from thread 1 of 5
Hello from thread 3 of 5
Hello from thread 0 of 5
Hello from thread 4 of 5
pi@raspberrypi:~ $ ./spmd2 8
Hello from thread 2 of 8
Hello from thread 1 of 8
Hello from thread 0 of 8
Hello from thread 3 of 8
Hello from thread 4 of 8
Hello from thread 7 of 8
Hello from thread 5 of 8
Hello from thread 6 of 8

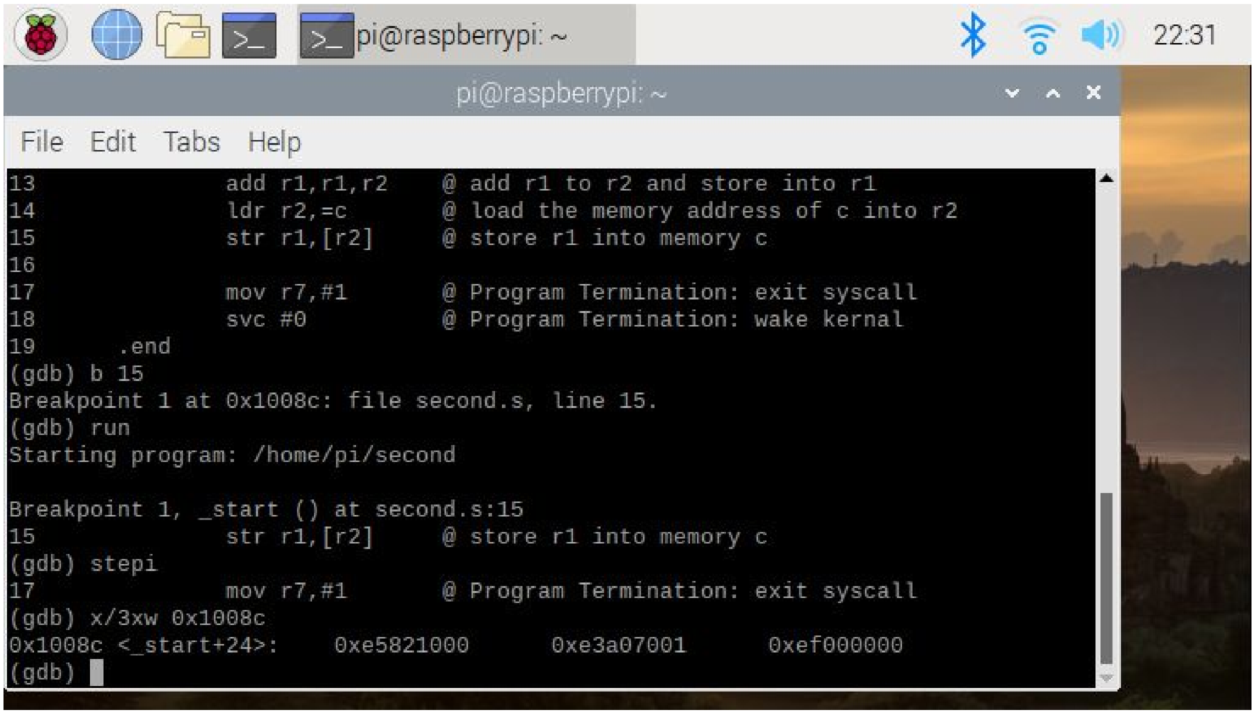

pi@raspberrypi: ~
22:06
pi@raspberrypi: ~
File Edit Tabs Help
GNU nano 3.2 spmd2.c
#include <stdio.h>
#include <omp.h>
#include <stdlib.h>
int main(int argc, char** argv){
//int id, numThreads;
printf("\n");
if(argc>1){
omp_set_num_threads(atoi(argv[1]));
}
#pragma omp parallel
{
int id = omp_get_thread_num();
int numThreads = omp_get_num_threads();
printf("Hello from thread %d of %d\n",id,numThreads);
^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



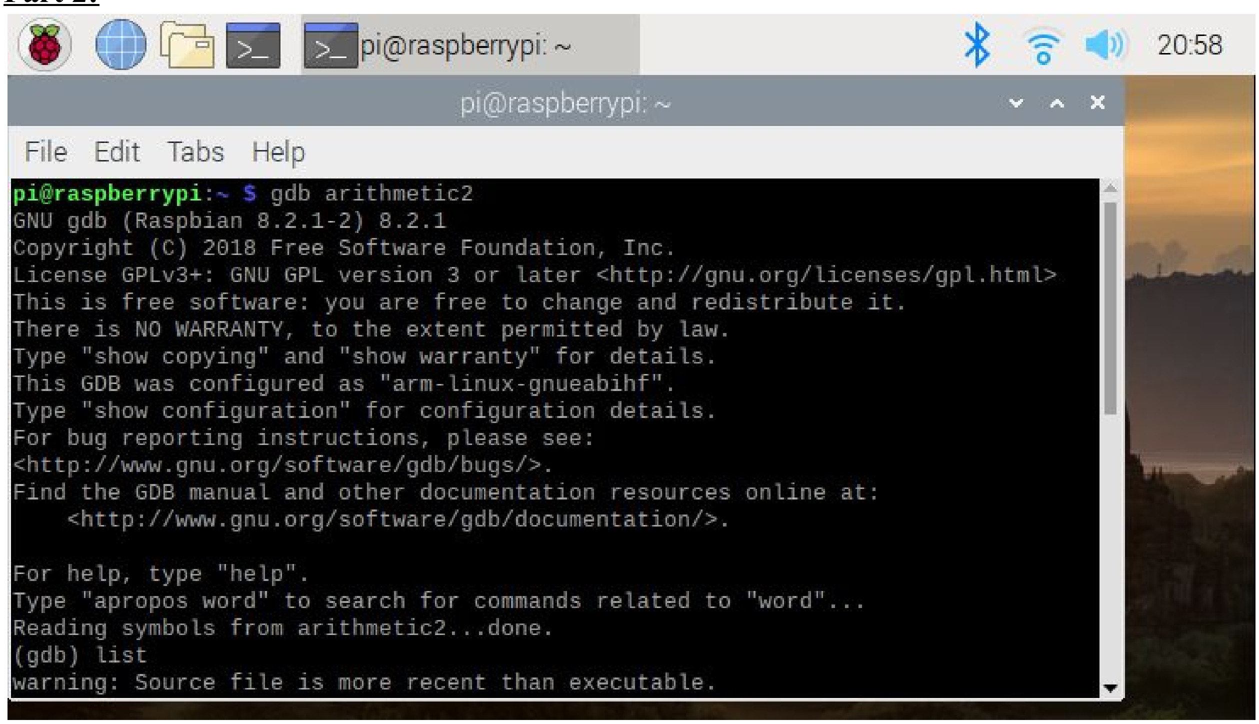
ARM Assembly Programming: Andre Nguyenphuc:

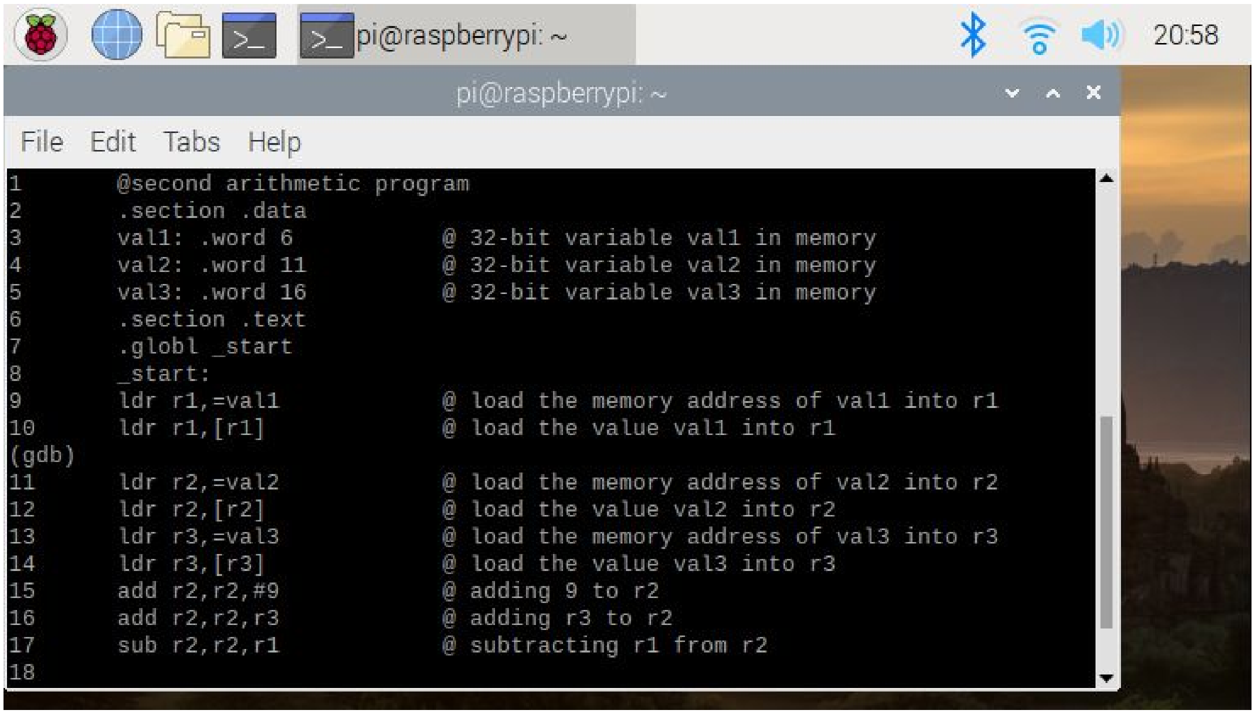

pi@raspberrypi: ~
22:31
pi@raspberrypi: ~
File Edit Tabs Help
pi@raspberrypi:~ $ gdb second
GNU gdb (Raspbian 8.2.1-2) 8.2.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and 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-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 second...done.
(gdb) list
1 @ second program: c = a + b

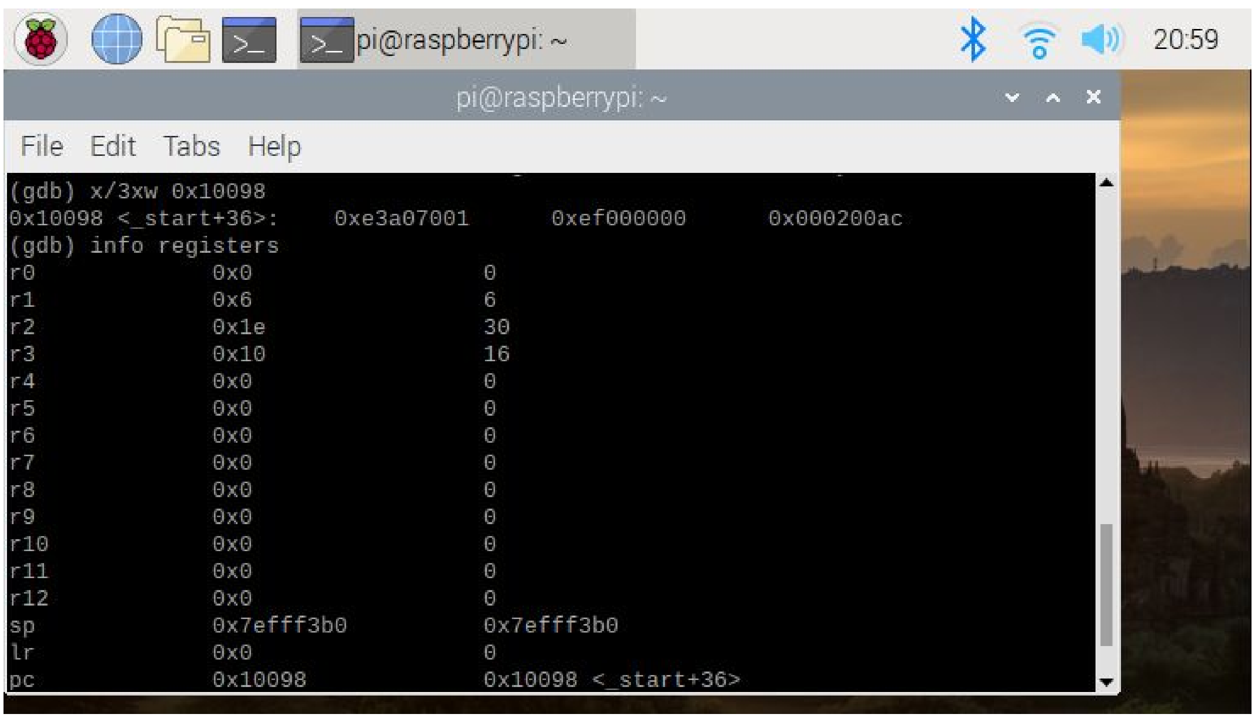

pi@raspberrypi: ~
22:31
pi@raspberrypi: ~
File Edit Tabs Help
1 @ second program: c = a + b
2 .section .data
3 a: .word 2 @ 32-bit variable a in memory
4 b: .word 5 @ 32-bit variable b in memory
5 c: .word 0 @ 32-bit variable c in memory
6 .section .text
7 .globl _start
8 _start:
9 ldr r1,=a @ load the memory address of a into r1
10 ldr r1,[r1] @ load the value a into r1
(gdb) list
11 ldr r2,=b @ load the memory address of b into r2
12 ldr r2,[r2] @ load the value b into r2
13 add r1,r1,r2 @ add r1 to r2 and store into r1
14 ldr r2,=c @ load the memory address of c into r2
15 str r1,[r2] @ store r1 into memory c
16
17 mov r7,#1 @ Program Termination: exit syscall
18 svc #0 @ Program Termination: wake kernal


pi@raspberrypi: ~
22:31
pi@raspberrypi: ~
File Edit Tabs Help
13 add r1,r1,r2 @ add r1 to r2 and store into r1
14 ldr r2,=c @ load the memory address of c into r2
15 str r1,[r2] @ store r1 into memory c
16
17 mov r7,#1 @ Program Termination: exit syscall
18 svc #0 @ Program Termination: wake kernal
19 .end
(gdb) b 15
Breakpoint 1 at 0x1008c: file second.s, line 15.
(gdb) run
Starting program: /home/pi/second
Breakpoint 1, _start () at second.s:15
15 str r1,[r2] @ store r1 into memory c
(gdb) stepi
17 mov r7,#1 @ Program Termination: exit syscall
(gdb) x/3xw 0x1008c
0x1008c <_start+24>: 0xe5821000 0xe3a07001 0xef000000
(gdb)

Part 2:


pi@raspberrypi: ~
20:58
pi@raspberrypi: ~
File Edit Tabs Help
pi@raspberrypi:~ $ gdb arithmetic2
GNU gdb (Raspbian 8.2.1-2) 8.2.1
Copyright (C) 2018 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-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 arithmetic2...done.
(gdb) list
warning: Source file is more recent than executable.

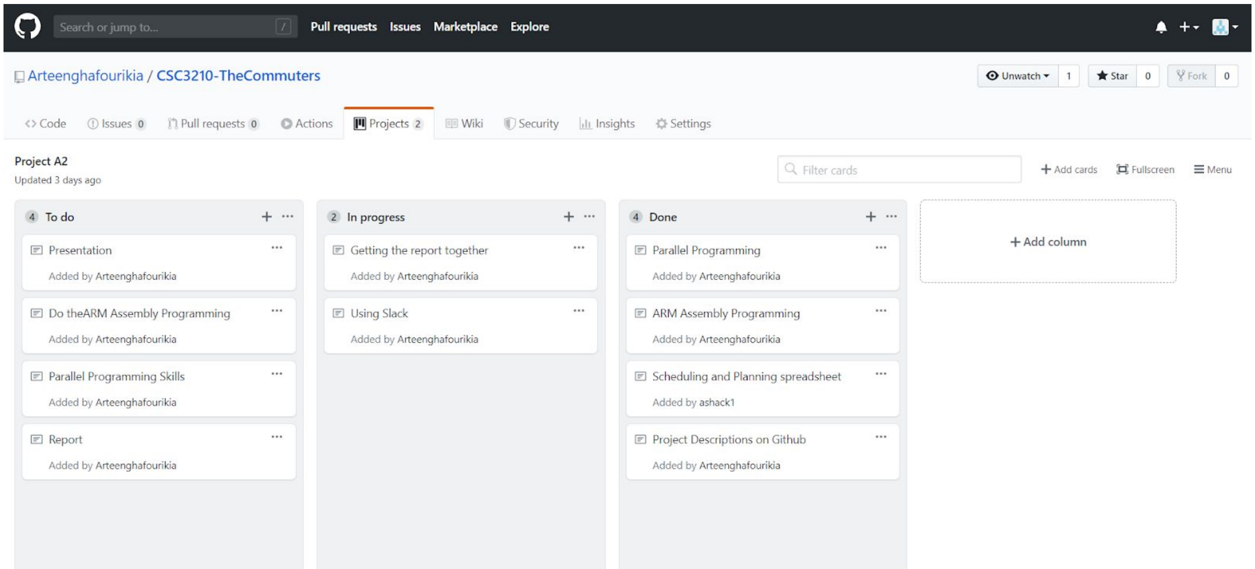




Appendix

Slack: https://app.slack.com/client/TSWLWS9LK/CT8581ZU0

Github: https://github.com/Arteenghafourikia/CSC3210-TheCommuters
Youtube:


Search or jump to...
Pull requests
Issues
Marketplace
Explore
Arteenghafourikia / CSC3210-TheCommuters
Unwatch 1
Star 0
Fork 0
Code
Issues 0
Pull requests 0
Actions
Projects 2
Wiki
Security
Insights
Settings
Project A2
Updated 3 days ago
Filter cards
Add cards
Fullscreen
Menu
4 To do
Presentation
Added by Arteenghafourikia
Do theARM Assembly Programming
Added by Arteenghafourikia
Parallel Programming Skills
Added by Arteenghafourikia
Report
Added by Arteenghafourikia
2 In progress
Getting the report together
Added by Arteenghafourikia
Using Slack
Added by Arteenghafourikia
4 Done
Parallel Programming
Added by Arteenghafourikia
ARM Assembly Programming
Added by Arteenghafourikia
Scheduling and Planning spreadsheet
Added by ashack1
Project Descriptions on Github
Added by Arteenghafourikia
+ Add column