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 the planning and scheduling table. Complete individual parallel programming skills task and ARM assembly programming task.	3 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.	1.5 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.	2 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.	4 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.	2 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.**
  - One main difference between CISC and RISC is the instruction set. CISC has a larger instruction set and a vast amount of options and more functional capabilities in its instruction set, which allows more complex instructions to access memory. In RISC, the instruction set contains 100 or less instructions. Therefore, RISC has more registers, but CISC has more operations, addressing modes, and less registers.
  - Another difference is that ARM architecture (after version 3) changed to BI-endian, which has a feature that allows you to switch between big endian and little endian, but X86 uses the little-endian format.
  - RISC uses instructions that operate only on registers and only load/store instructions can access memory, which means it typically takes more lines of code and instructions to complete the desired task. However, CISC allows for a reduction in the amount of code and instructions needed to perform the desired task because direct access to memory is allowed.
- **What is the difference between sequential and parallel computation and identify the practical significance of each?**
  - In sequential computation, instructions are executed one after another, and only one instruction may execute at any moment in time. However, with parallel computation, instructions can be executed at the same time on different processors at any moment.
  - Sequential/serial computation is what software has traditionally been written for, but sequential computation can be impractical for solving larger problems. This is where parallel computation is more efficient because it is able to decrease the time needed to solve the problem because more resources are used at once to solve the problem.
- **Identify the basic form of data and task parallelism in computational problems.**
  - Data parallelism is when there are multiple data items, and the same computation is performed to the items. In this form, the data will be divided up among multiple processors and each processor will perform the same computation.
  - Task parallelism is when there are multiple data items, but there are different computations or "tasks" needed to be performed to the items. In this form, the data will be divided up among processors, and each processor will perform a different computation.
- **Explain the differences between processes and threads.**

- One main difference between processes and threads are that processes do not share memory with each other, while threads share the common memory of the process that they belong to.
- Also, a process is an abstraction of a program in execution. However, a thread is a type of process that is said to be a lightweight process that allows a process to be decomposed into smaller parts.

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

- OpenMP is a standard that compilers who implement it must adhere to, and it provides support for parallel-programming in environments where memory is shared. In OpenMP, the library handles thread creation and management to make the programmer's life easier, by making their work less error-prone.
- OpenMP pragmas are compiler directives that enable the compiler to generate threaded code.

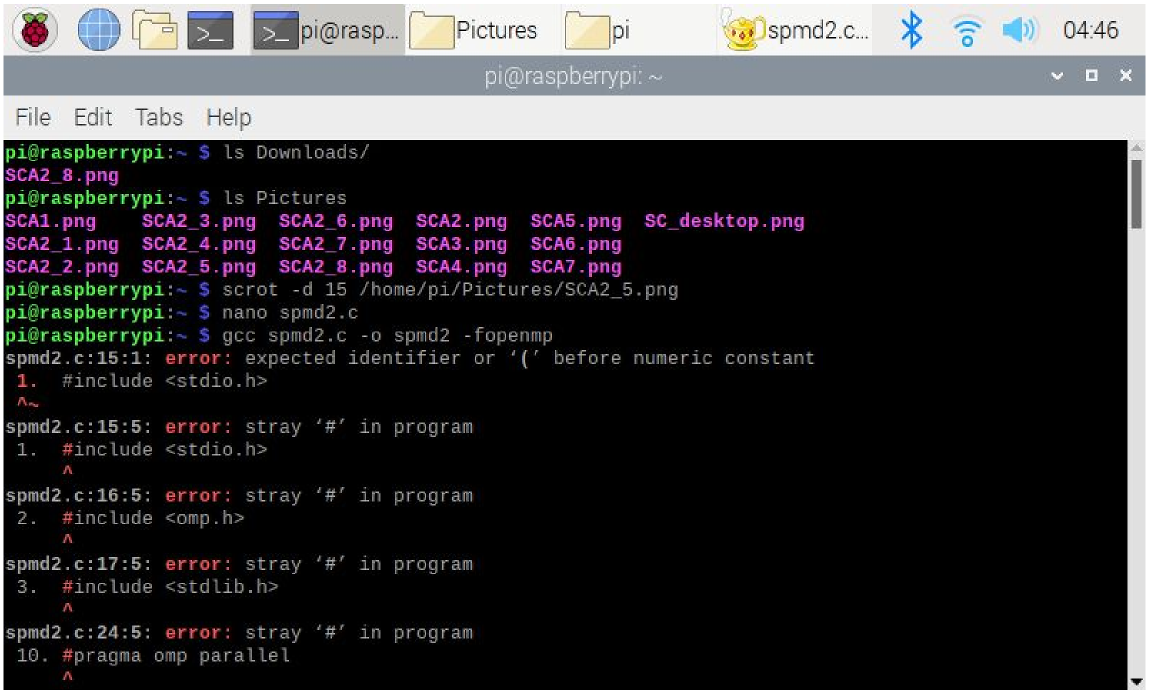
➔ **What applications benefit from multi-core (list four)?**

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

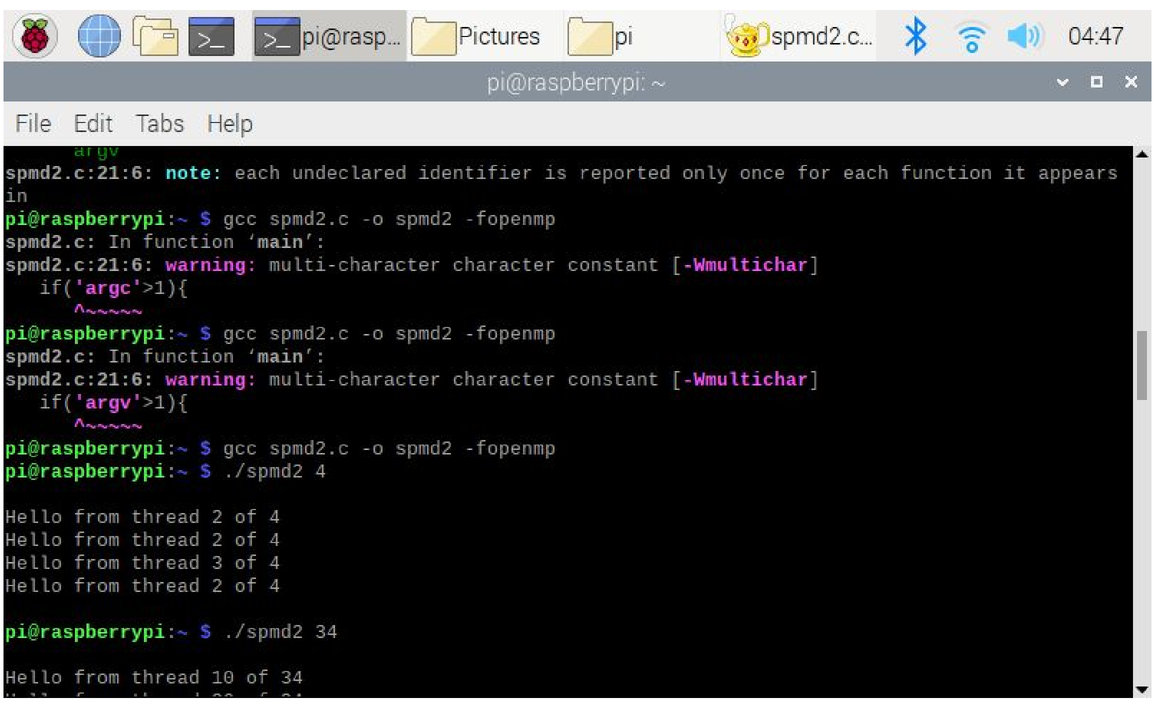
➔ **Why Multicore? (why not single core, list four)**

- 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, thus one should want to keep with current times.
  - The more cores that a CPU has results in the OS executing more processes at once, which increases the throughput of a system.
  - Multi-cores increase the clock speed of the processor without having as many heat problems that single cores have when trying to increase their clock speed.

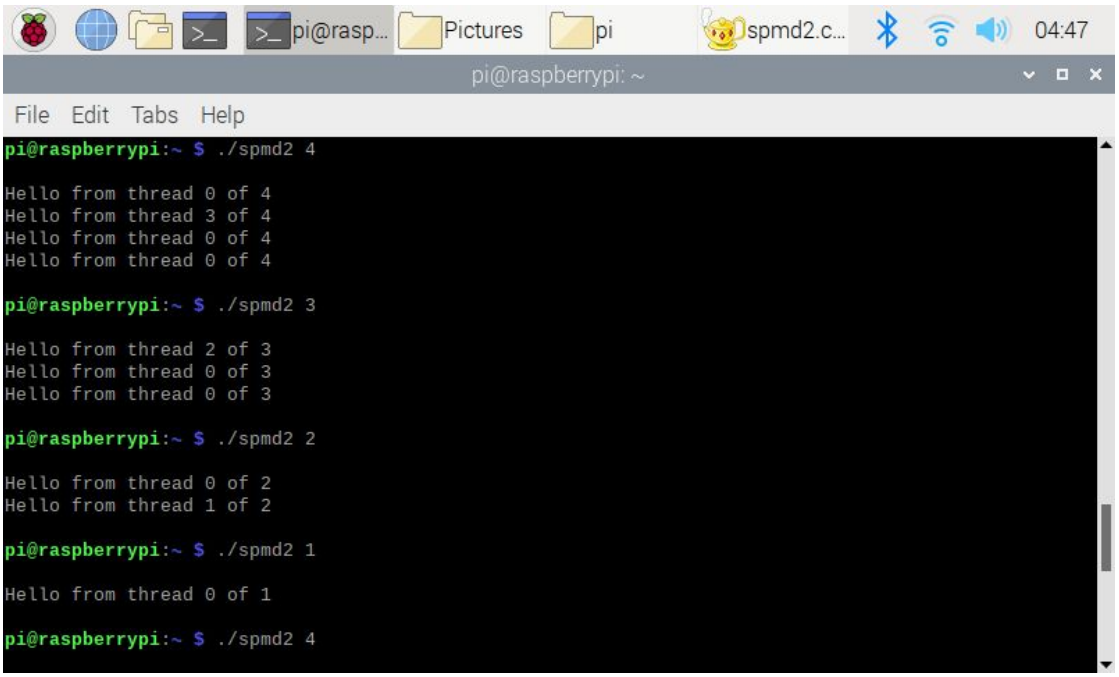
Parallel Programming Basics: Alaya Shack



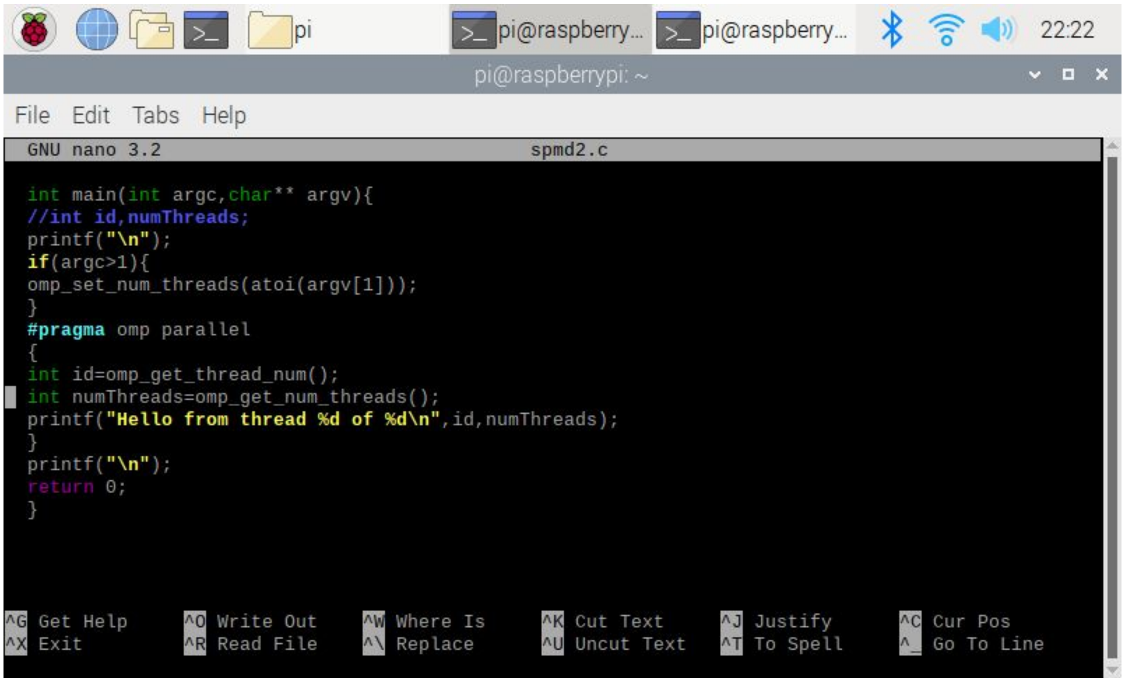
This screenshot uses the terminal trick, tab completion of long names. I tried the trick with ls Dow[Tab], and I tried it with ls Pic[Tab]. It filled in the remaining part of the directories, and it listed all the items that belong in that directory. Also, in this screenshot, I used "gcc spmd2.c -o spmd2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



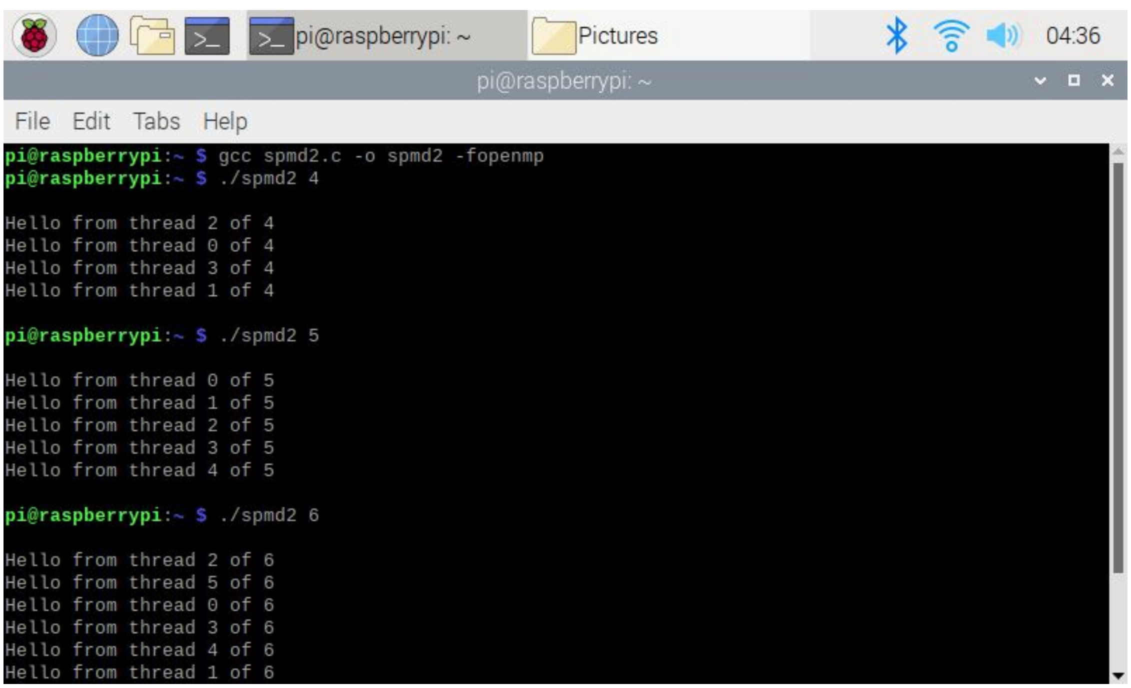
In this screenshot, I typed the same command to make the program executable. After I fixed my errors, I was finally able to make the spmd2.c an executable program. Then, I typed "./spmd2 4" to run the program.



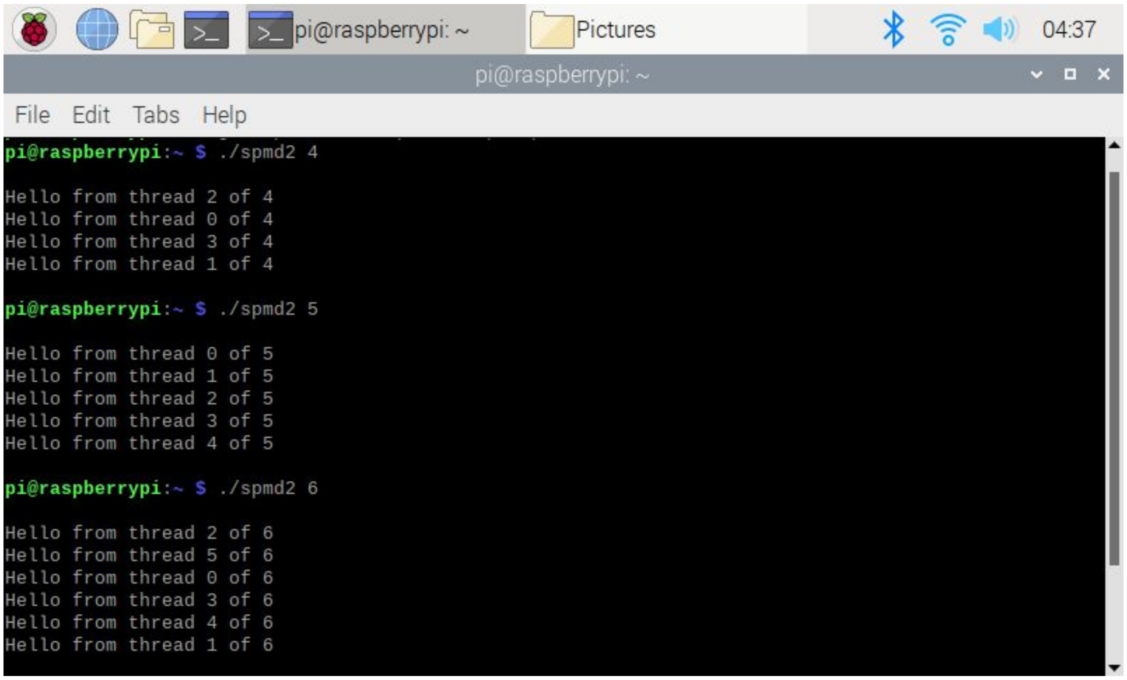
In this screenshot, I kept typing "./spmd2 4", but I alternated the 4 with 3,2, and 1. I learned that the 4 is a command-line argument that indicates how many threads to fork and that it can be changed. When I changed the argument, I noticed that the message printed the amount of times that the argument 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.

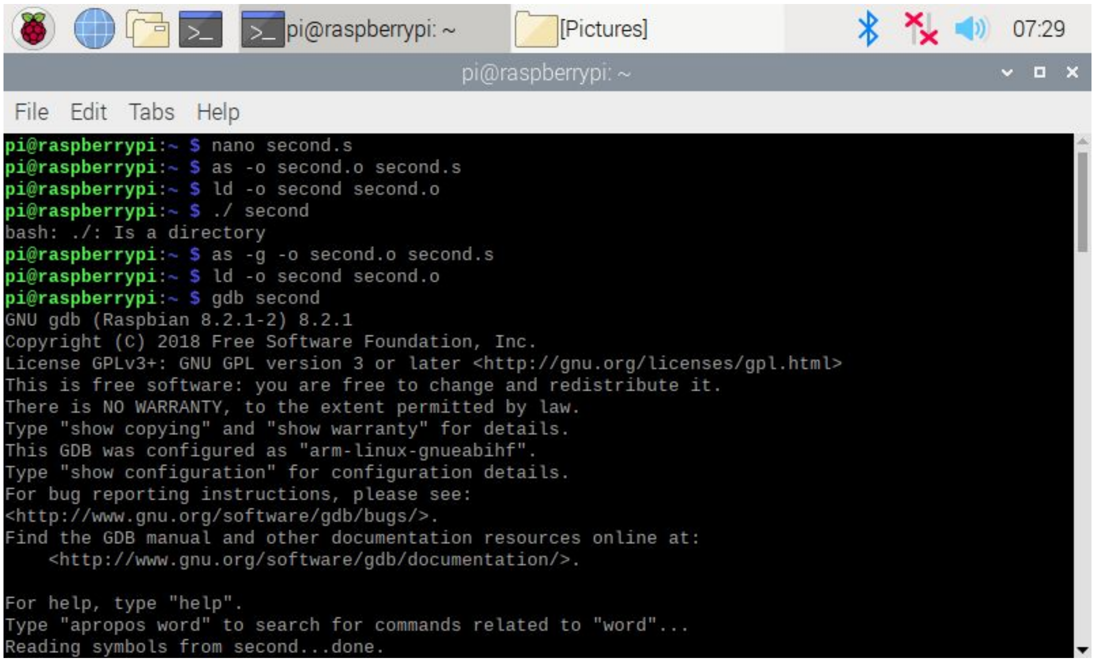


In this screenshot, I made the changes to the code. I typed "gcc spmd2.c -o spmd2 -fopenmp" to compile the program, and I ran the program with "./spmd2 4".

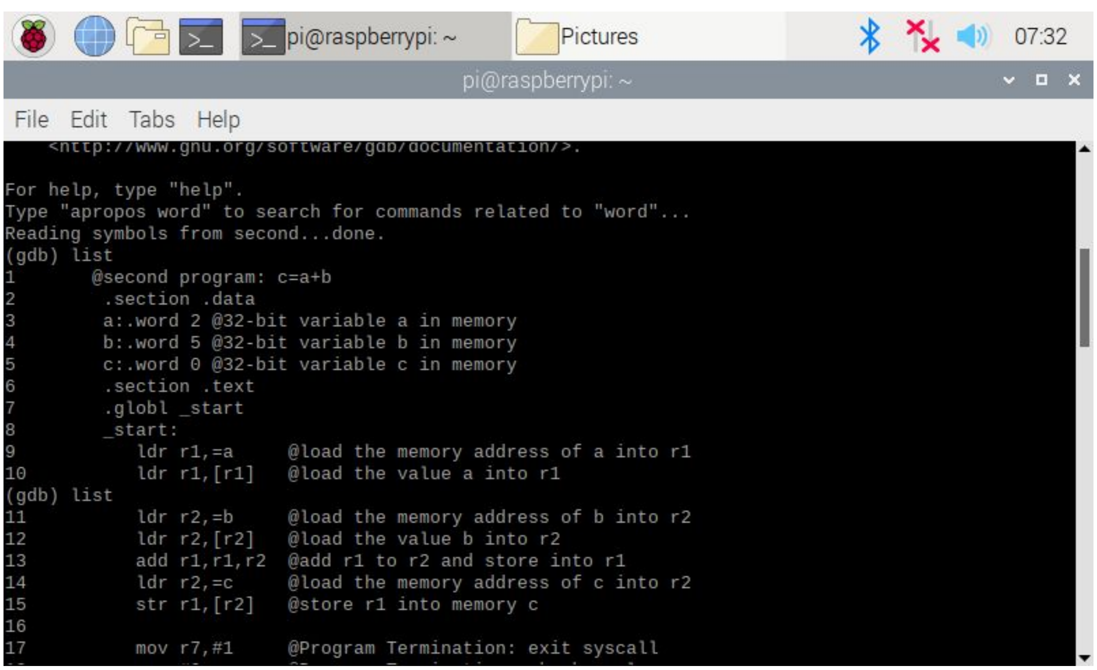


In this screenshot, I alternated the command-line argument 4 with 5 and 6 to make sure that the program was running correctly. The program was running correctly because now the thread id number only appears once. Although the thread id numbers do not print in order, I learned that this is okay because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

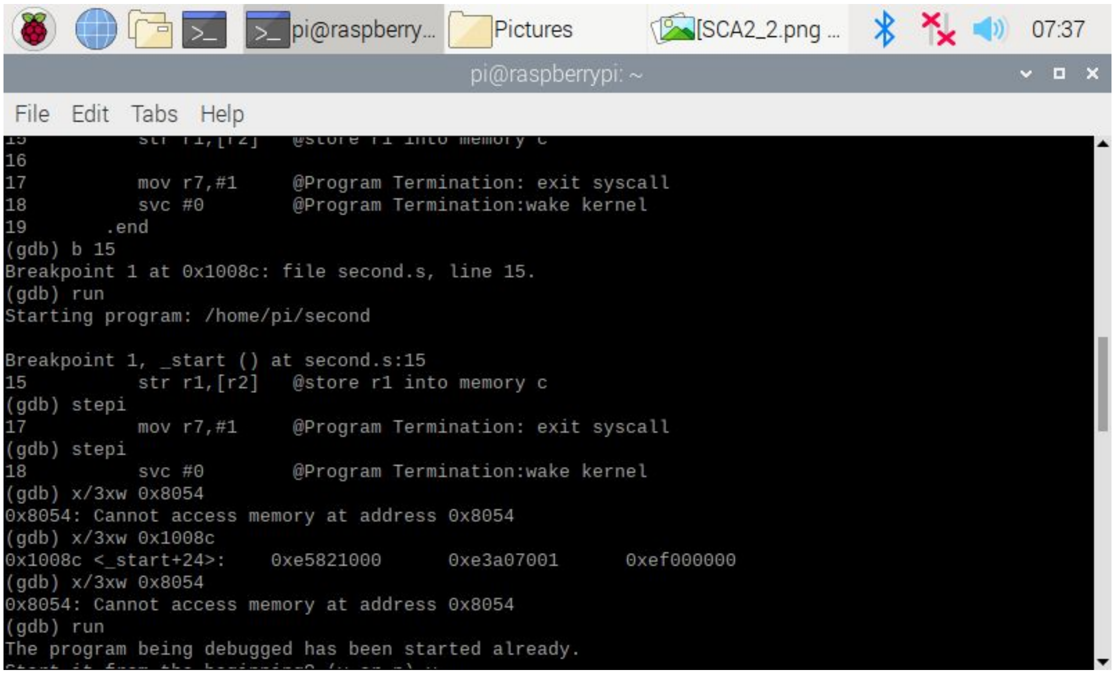
Arm Assembly Programming:Alaya Shack



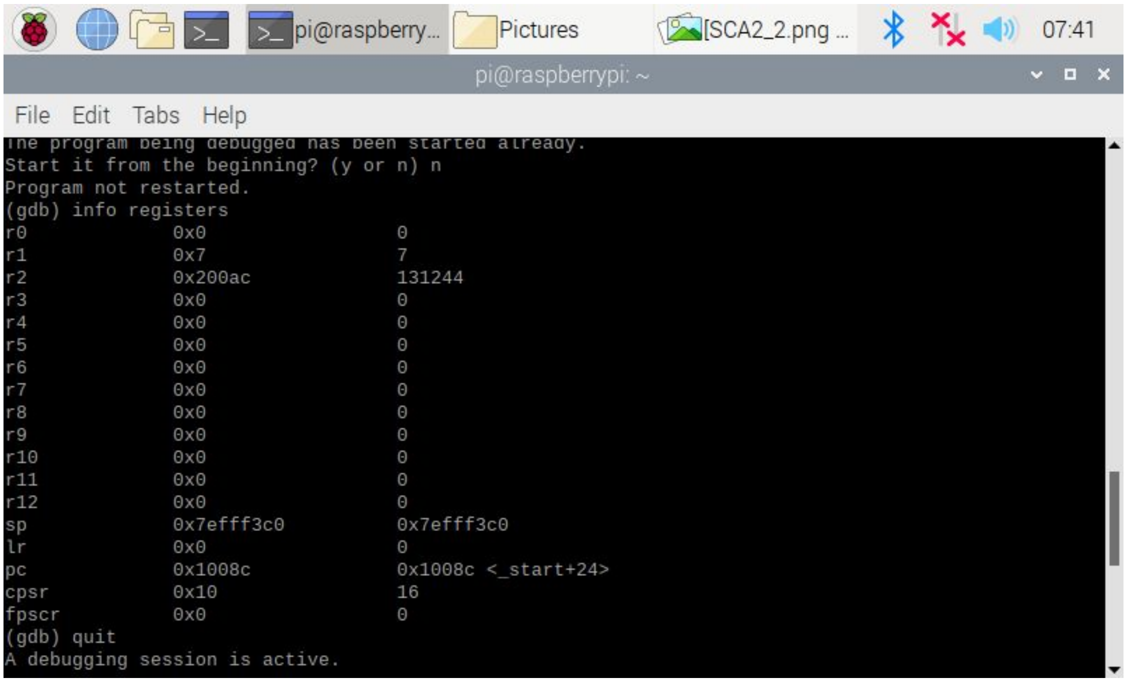
This screenshot shows how I started the “second” file with “nano second.s”. Once, I finished writing the file I was able to assemble and link the second program. I typed “./ second” to run the program, as directed by the assignment, and the “bash: ./: Is a directory” message appeared. However, I tried 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.



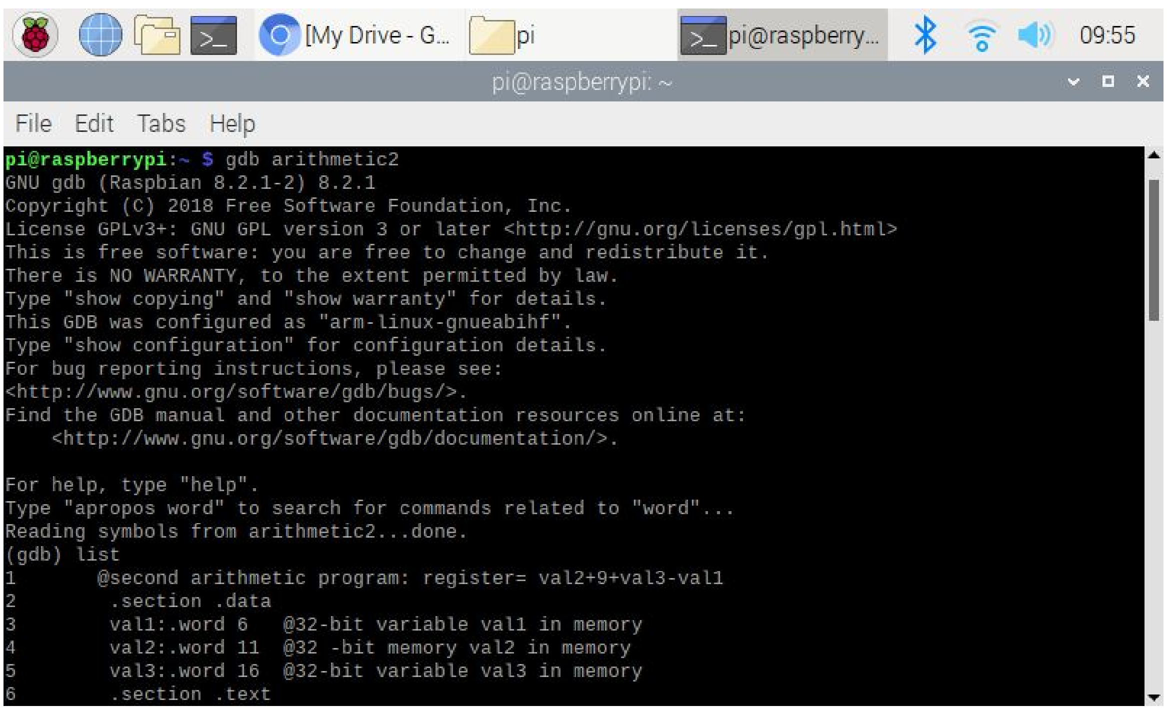
In this screenshot, I used the “gdb list” command to list the first ten lines of code, and I repeated the command and listed the next ten lines of code for the second program.



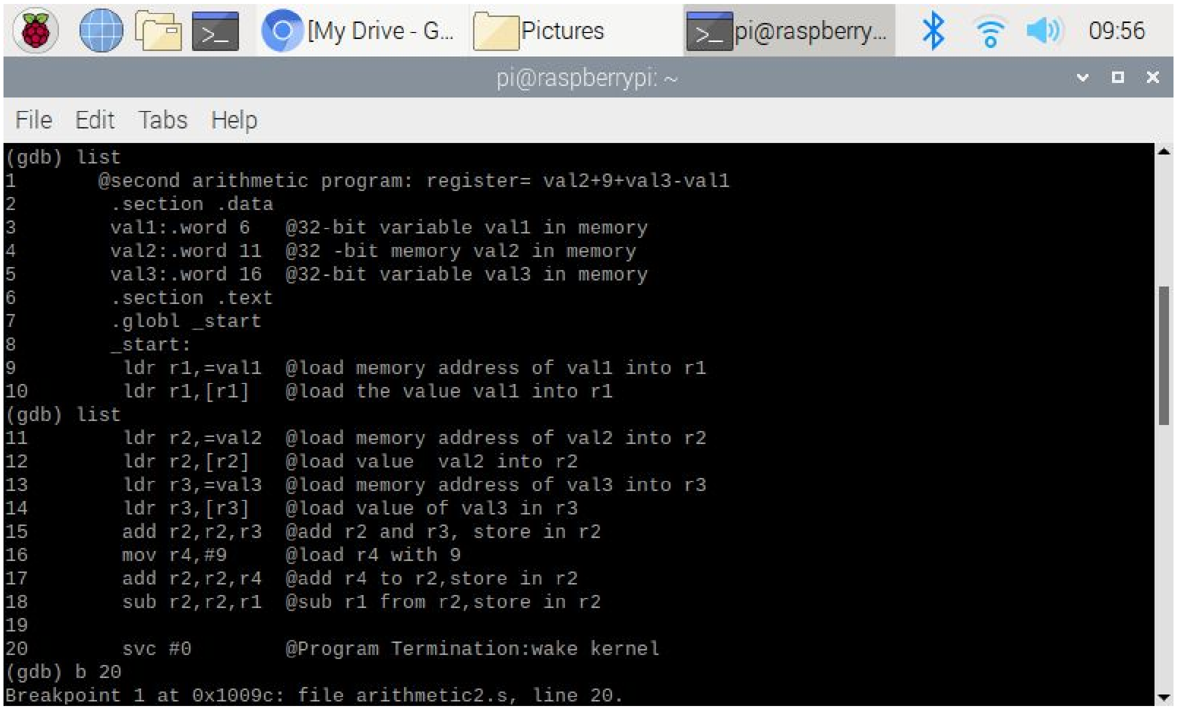
This screenshot shows where I inserted a breakpoint at line 15. After I inserted the breakpoint, I noticed that the next line shows the memory address of where I inserted the breakpoint. Also, in this screenshot, I started the debugging process with “gdb run”. Then, I used “gdb stepi” to step through the program one line at a time. Next, I began 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.



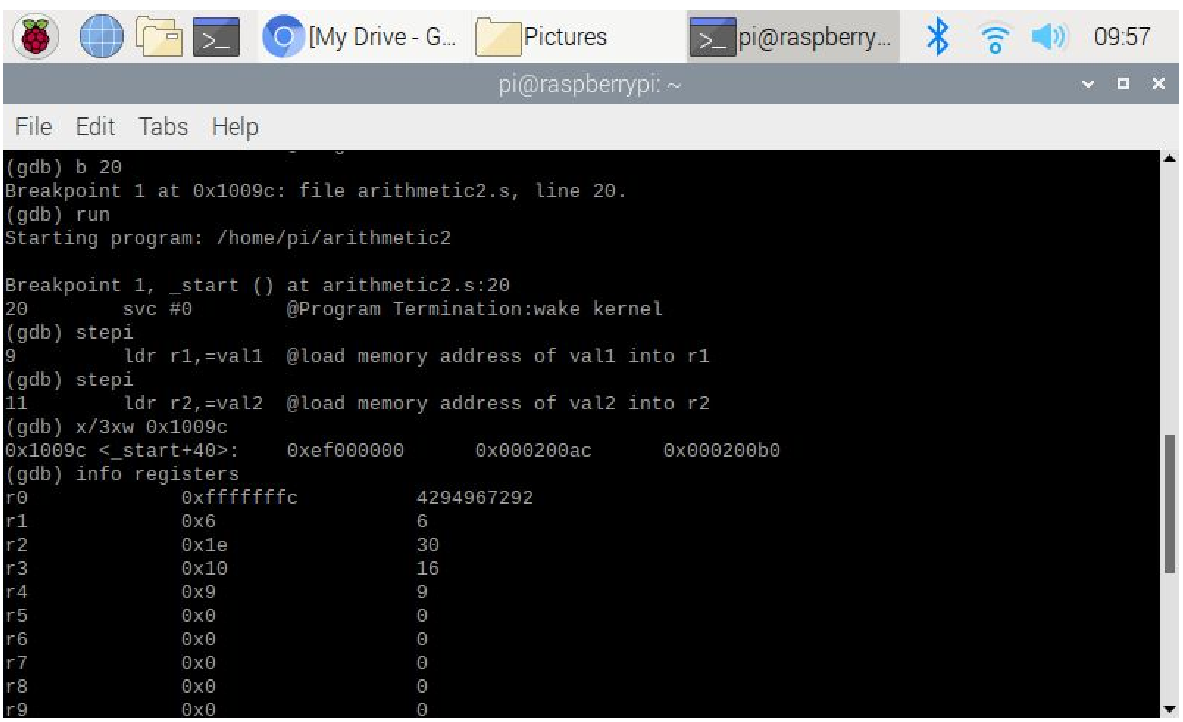
In this screenshot, I used “gdb info registers” to examine the registers. In r1, 7 is there because 2(r1) and 5(r2) are added together and stored in r1. In r2, the memory address of c is loaded there.



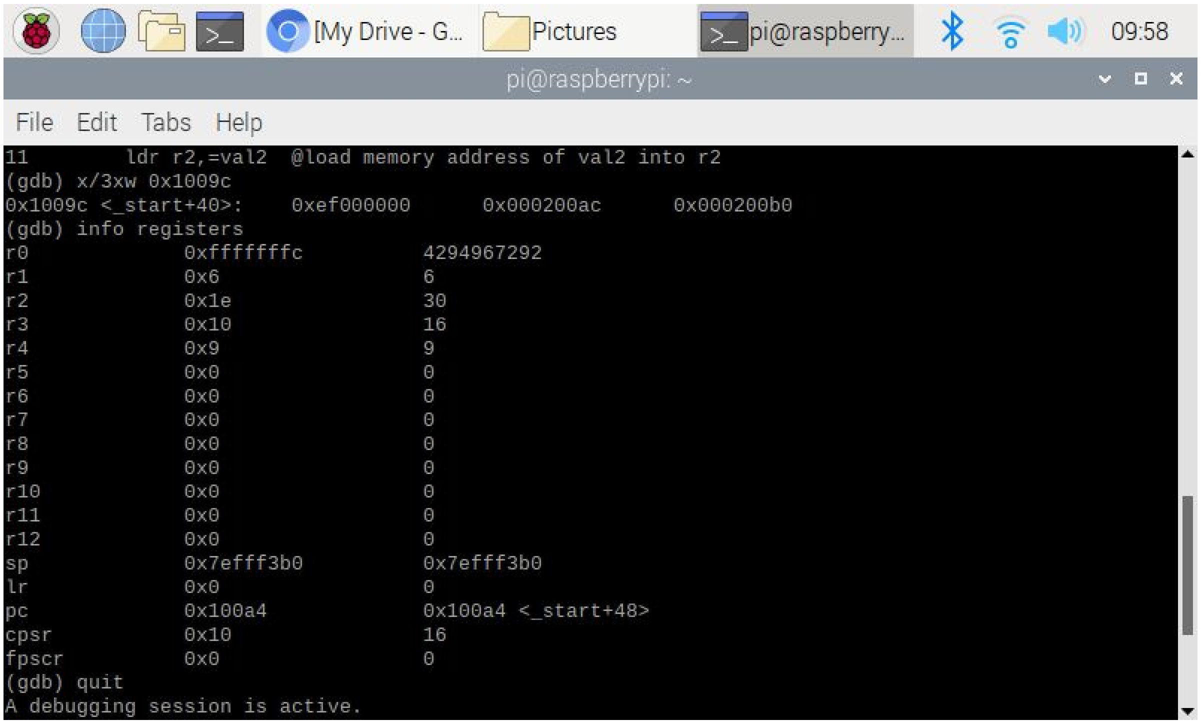
This screenshot shows the launch of the debugger with “gdb arithmetic2”.



In this screenshot, I listed the first ten lines of code with “gdb list”, and I listed the next ten lines of code by repeating that same command.



In this screenshot, I set the breakpoint at line 20, and the memory address at which I placed the breakpoint appears “0X1009c”. Also, I ran the debugger, and then I used the “gdb stepi” to step through the program twice. Then, I used “gdb x/3xw 0x1009c” to examine the memory. Then three memory addresses appeared, which is where I assume the val1, val2, and val3 memory addresses are located.



In this screenshot, I used “gdb info registers” to access the registers. In r1, there is a 6 because I loaded the value 6 here. In r2, there is a 30 because I added 11(r2) and 16(r3), and I also added 9 to r2, which is 36. Then, I subtracted 6(r1) from 36(r2), which is 30 and in hex 30 is 1e. In r3, there is a 16 because I loaded the value 16 there. In r4, I used mov to load 9 here.

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

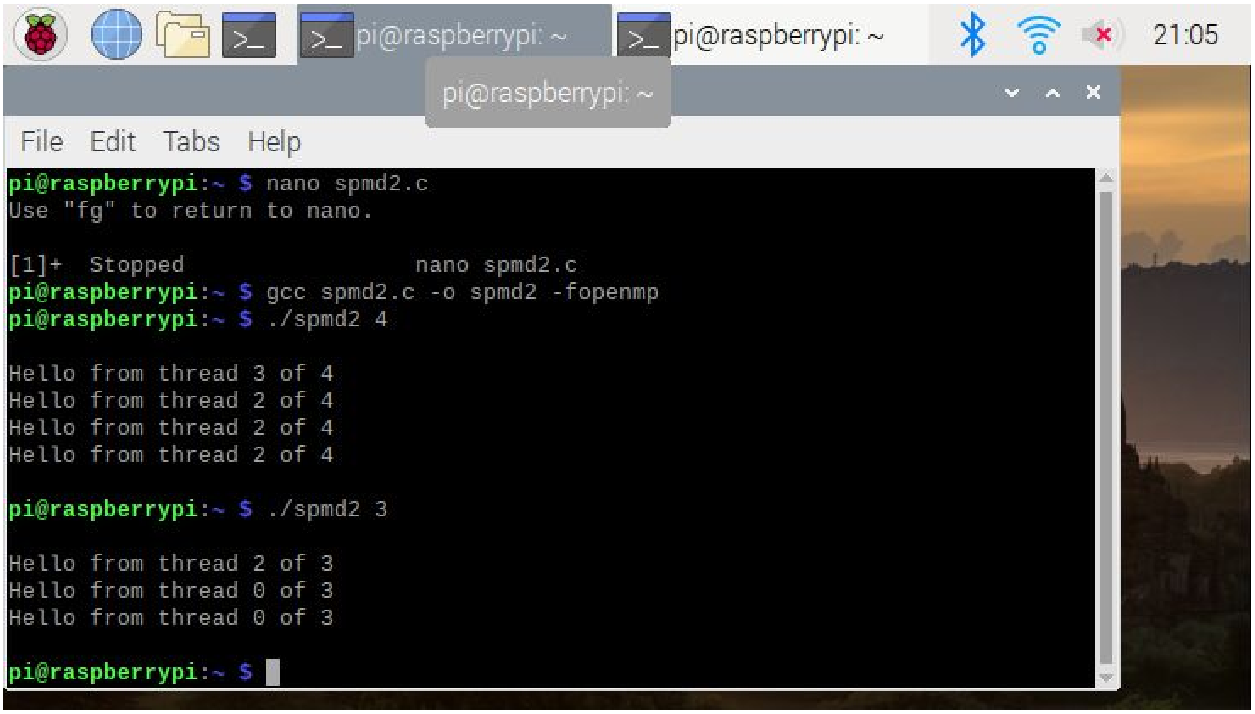
- **Identifying the components of the raspberry PI B+.**
  The components of the raspberry PI B+ are a single board computer with a Quad-core Multicore CPU with two USB ports, an ethernet controller, ethernet port, one GB RAM, camera port, two power ports, HDMI and the display ports.
- **How many cores 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.**
  - One of the differences between the ARM and INTEL processor is the instruction set. Intel has a CISC (Complex Instruction Set Computing) processor which allows for more versatile and detail oriented instructions to access memory while ARM has more registers.
  - ARM is a RISC (Reduced Instruction Set Computing) processor giving it a more simple instruction set which provides ARM more speed allowing instructions to be executed faster; however, it is more difficult to program efficiently.
  - Another notable difference is that ARM uses a BI-endian format after version 3, while Intel x86 still uses the little-endian format.
- **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. At the same time, parallel computing is broken down into a series of instructions where each part executes simultaneously on different processors. Parallel computing is more efficient as you can solve bigger problems faster, while you would use sequential computation when only using one thread.
- **Identify the basic form of data and task parallelism in computational problems.**
  - Data parallelism is when you perform the same computation to numerous data items where the data will be divided and computed by different processors
  - Task parallelism is when there are tasks that have to be done differently, and you give each task to a different process. The process will then be executed according to the given task and not the data.
- **Explain the differences between processes and threads.**
  - A process is the outline of a functioning program; they do not share the memory with other processes, While a thread is a process that can be broken into smaller parts and run individually. Threads also share the memory with the process they are in.
- **What is OpenMP and what is OpenMP pragmas?**
  - OpenMP is a library for parallel programming where it takes parts of the code and runs them parallel to the program.
  - OpenMP pragmas are the compilers that oversee how the program functions and makes sure the program doesn't run into any problems.
- **What applications benefit from multi-core (list four)?**
  - Compilers
  - Scientific applications

- Web Servers
- Multimedia applications

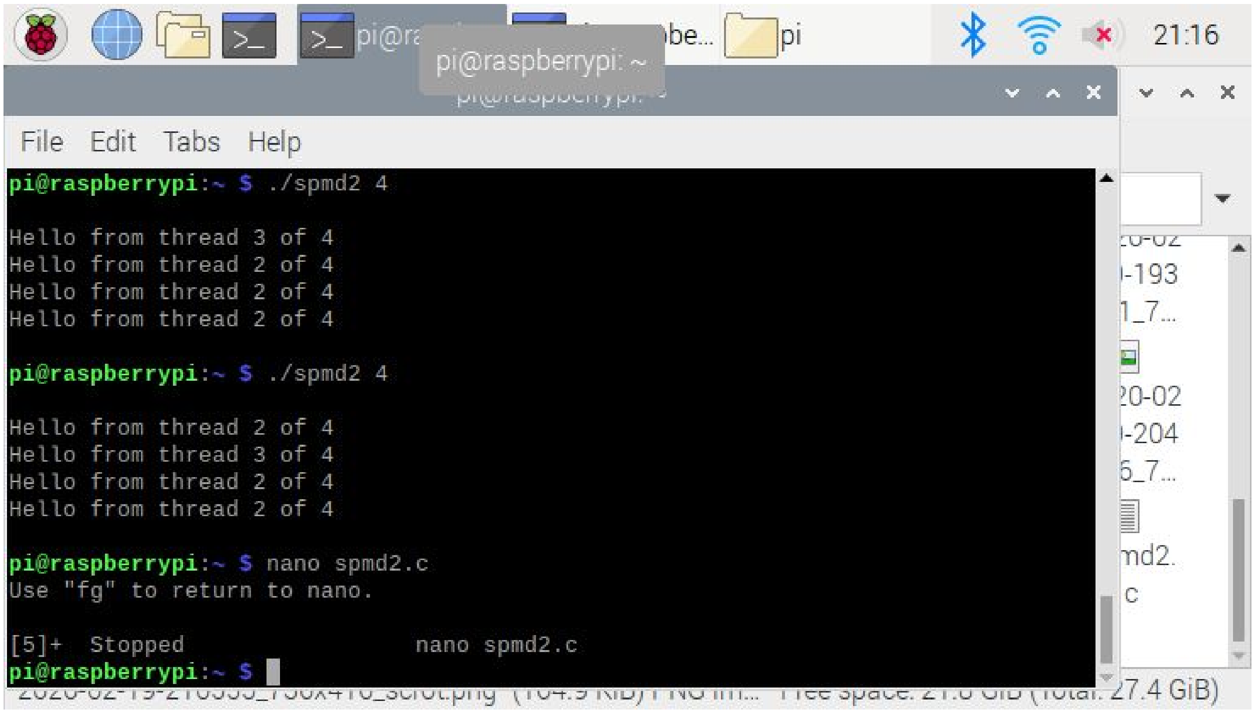
➔ **Why Multicore? (why not single core, list four)**

- Multicore is faster
- Computer architecture favors parallelism
- Single core has difficult design while Multicore has multiple processes making it more versatile
- Single core can overheat

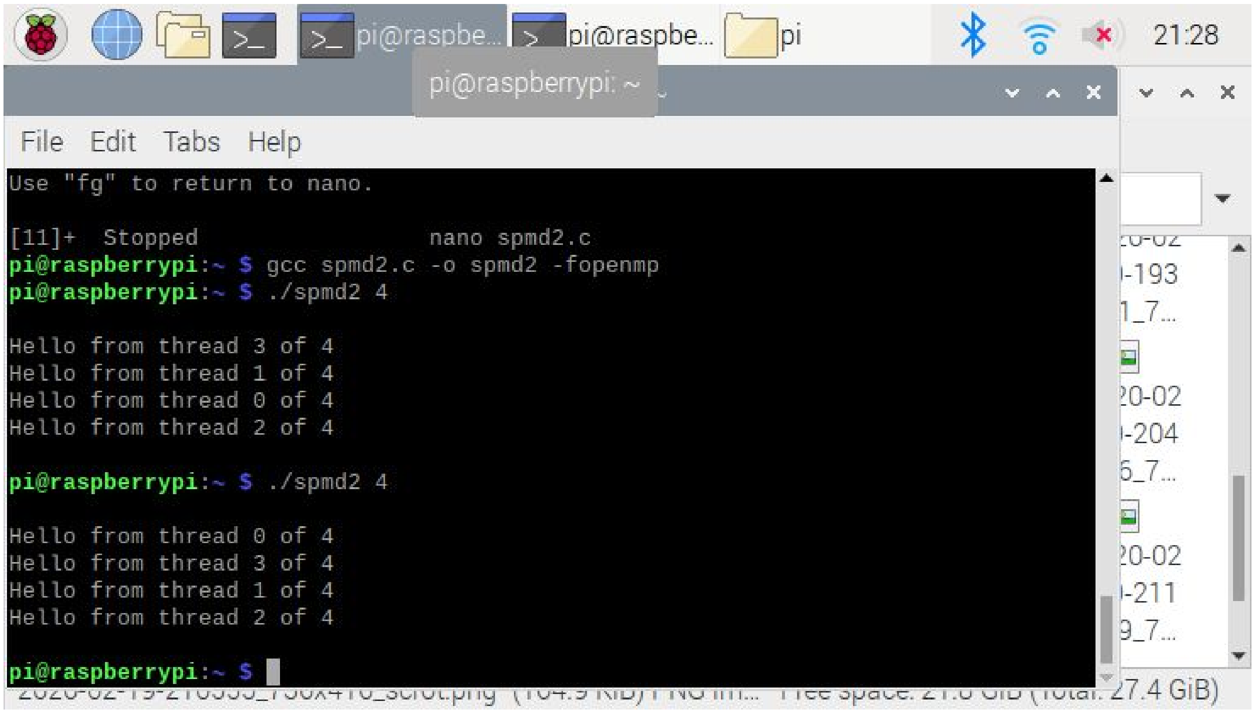
Parallel Programming Basics: Arteen Ghafourikia



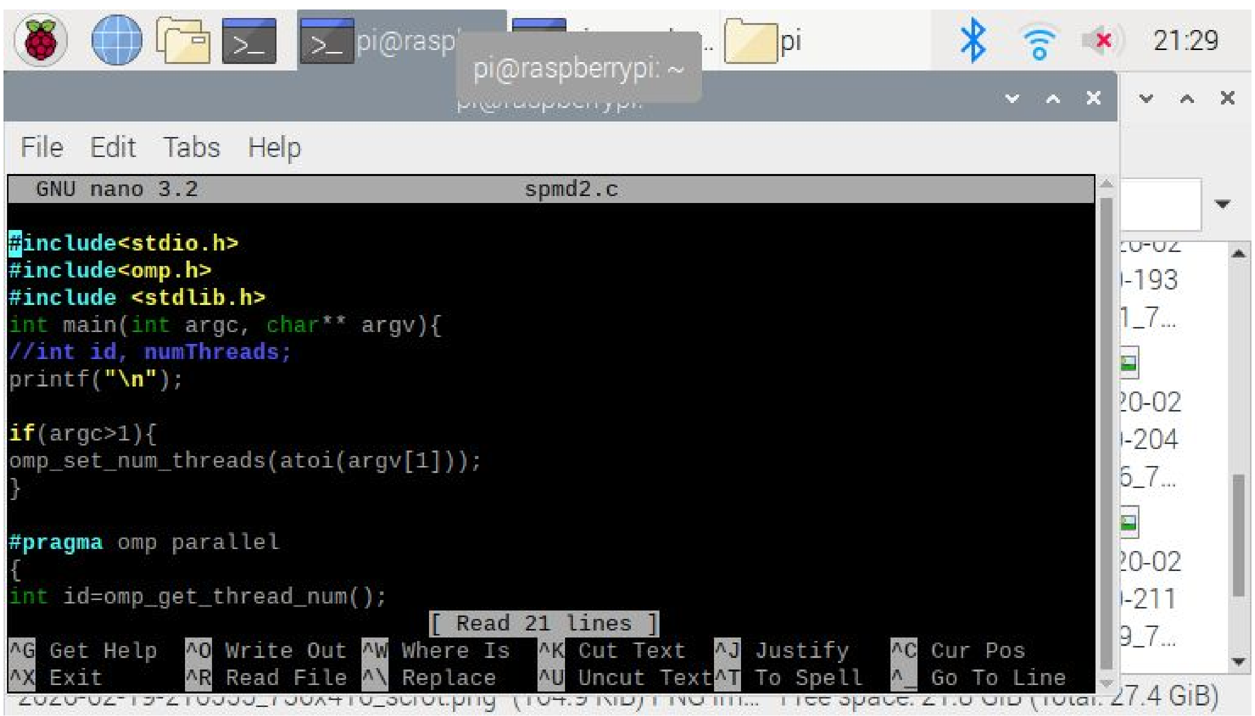
In this screenshot, I ran the C program that was given using ./spmd2 four and realized that the thread ids are all the same. I then alternated the number of threads, but still had threads that had the same id.

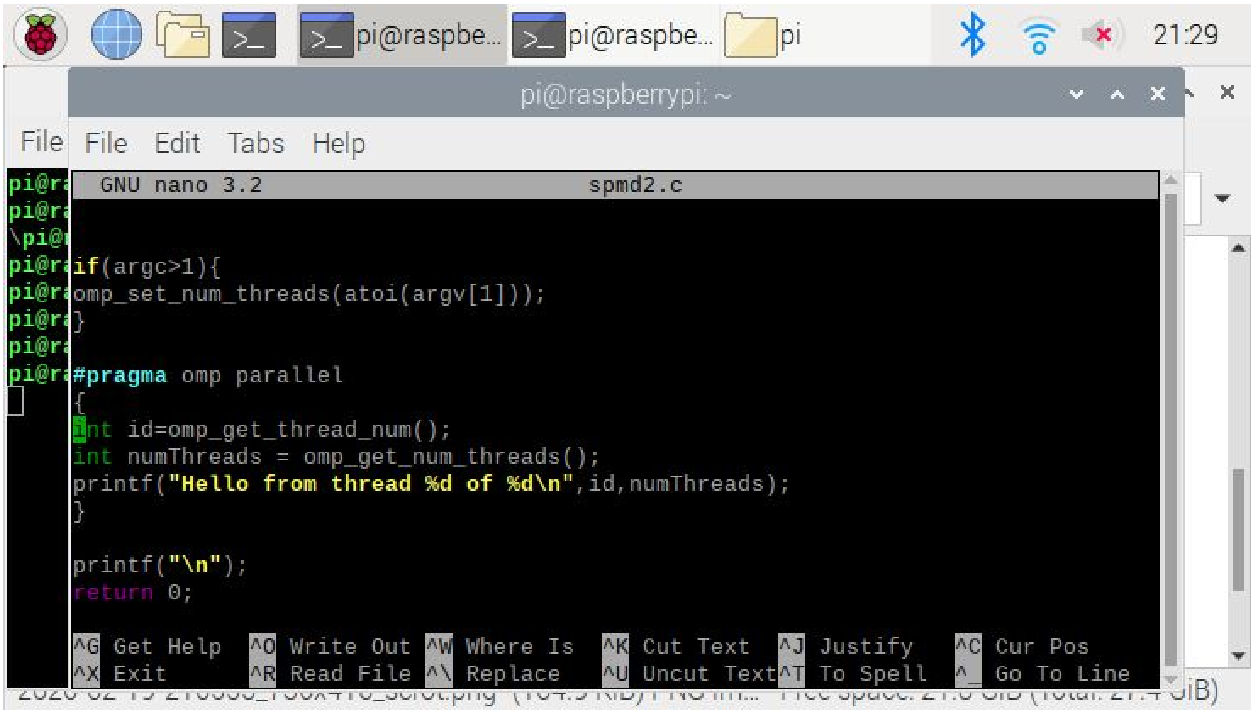


In this screenshot, I ran the program a couple more times to see if the thread ids would always be the same, and it turned out to be true.



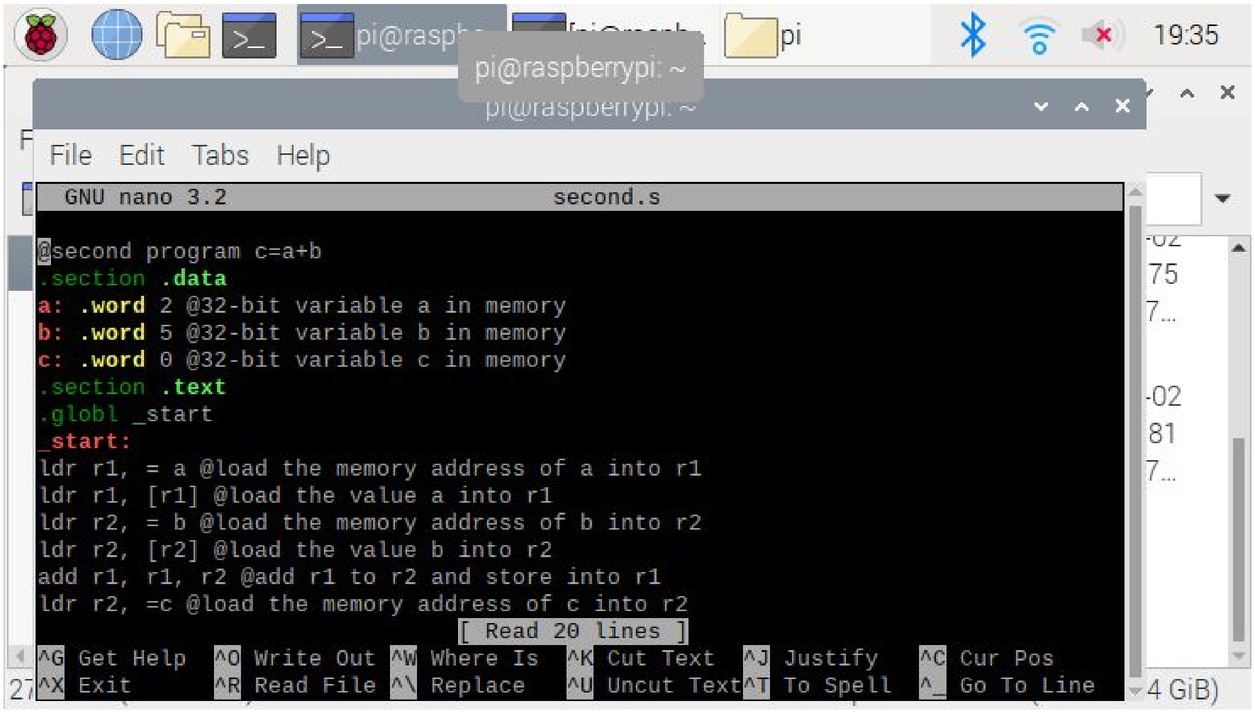
In this screenshot, I ran the program after I updated the code by declaring the variable within the block, so each thread keeps track of their id. The problem with the code was that it was only initializing the variable outside the block of code, causing the threads to share that variable's memory location. However, after I made the changes, the thread id displayed different values.



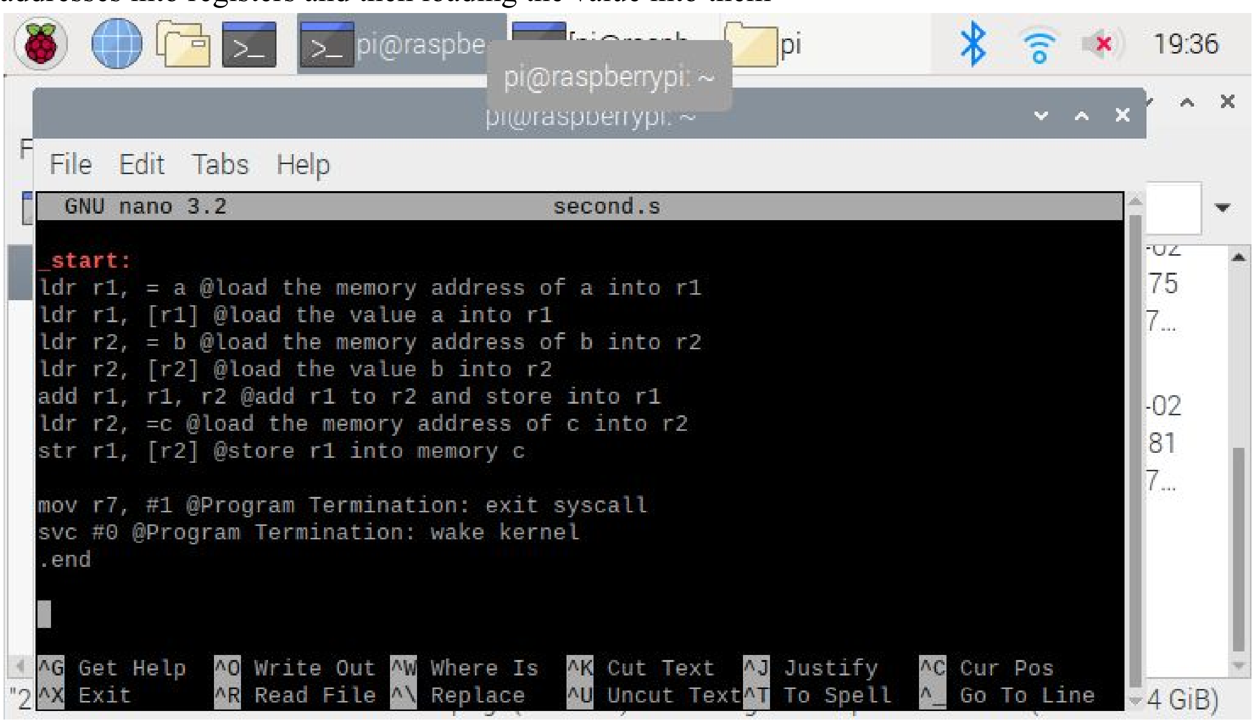


These two screenshots show the correct version of the code. The only difference between the correct version of the code and the old version is that the new version initializes a new variable giving us different thread ids. In contrast, the older one uses the same variable, sharing that variable's memory.

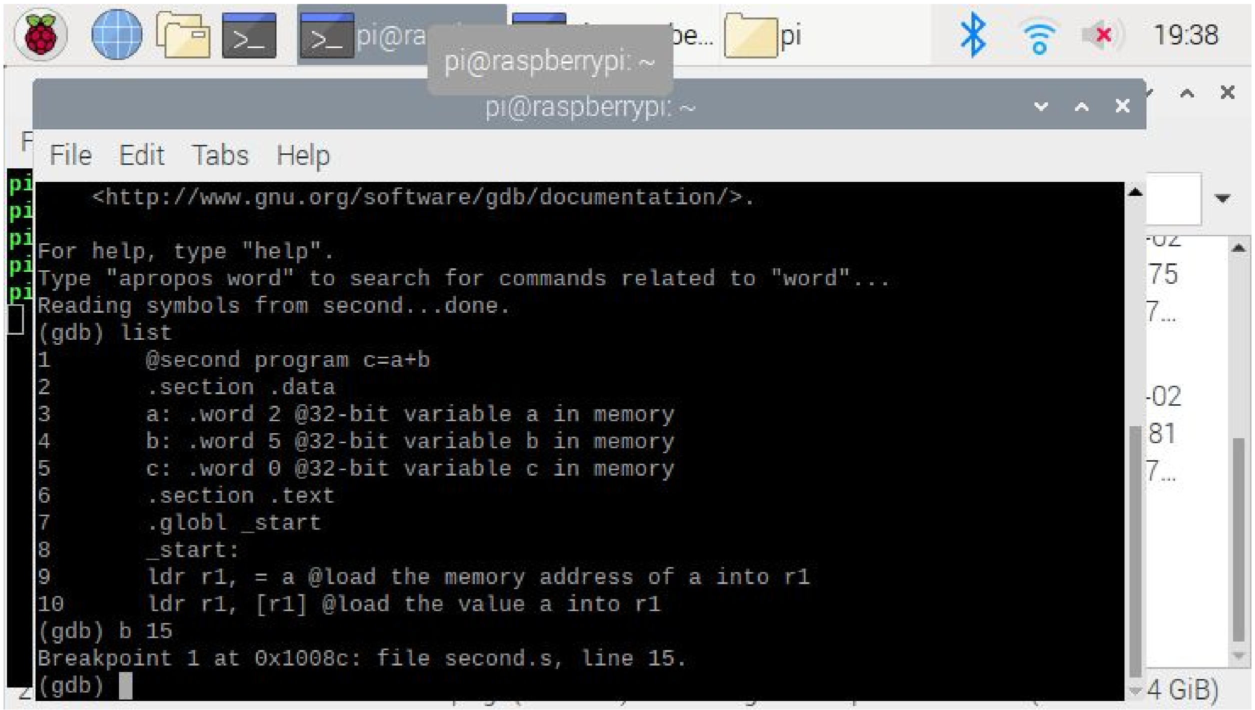
Arm Assembly Programming:Arteen Ghafourikia



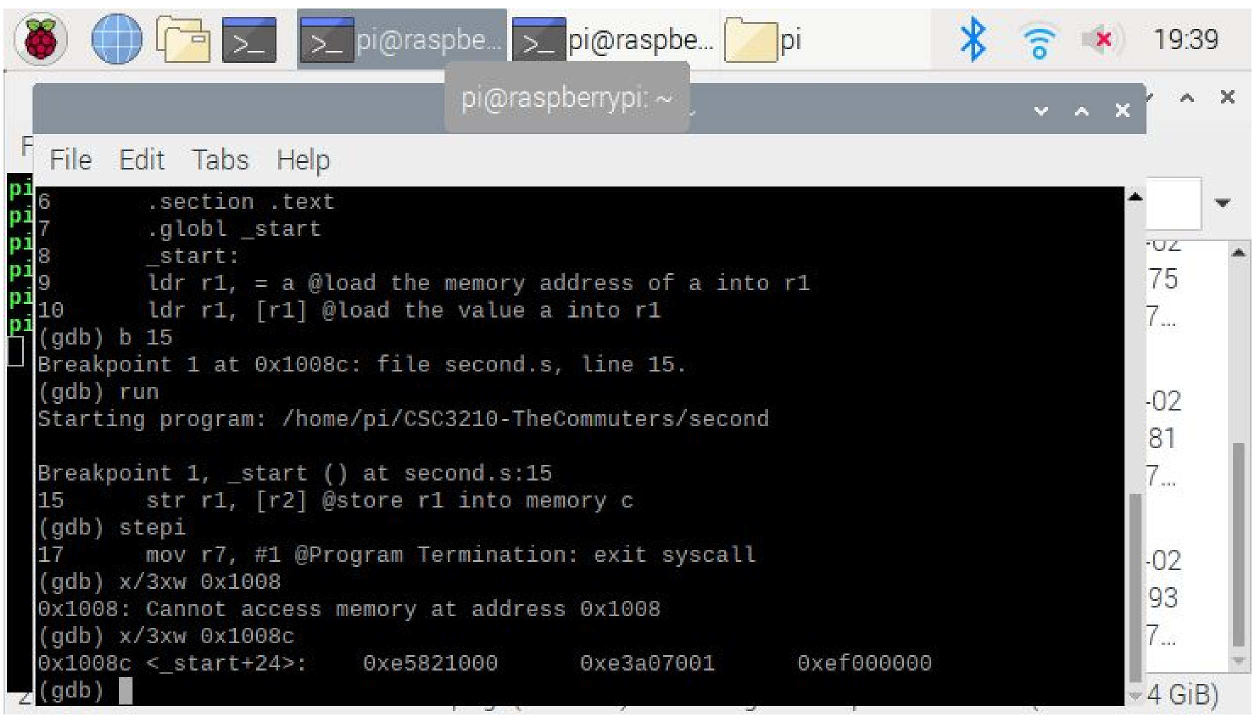
In this screenshot, I am showing the example code. In this code, we are loading the memory addresses into registers and then loading the value into them



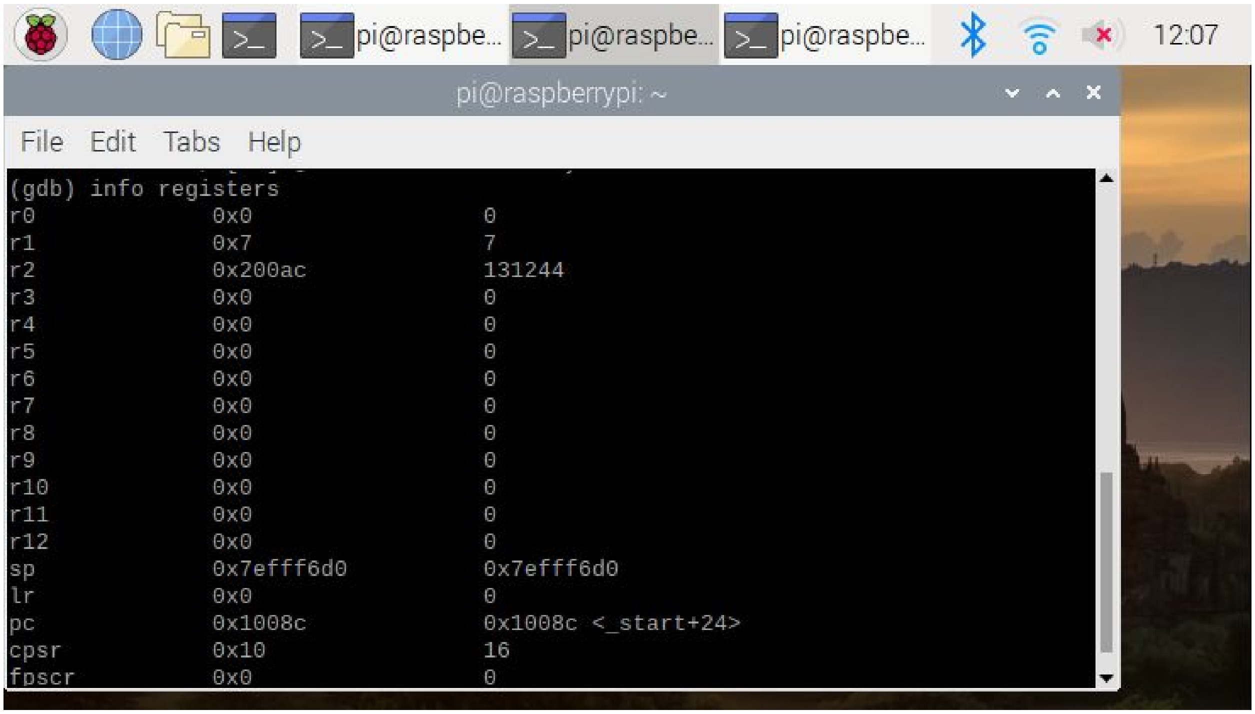
This screenshot is displaying the rest of the code.



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

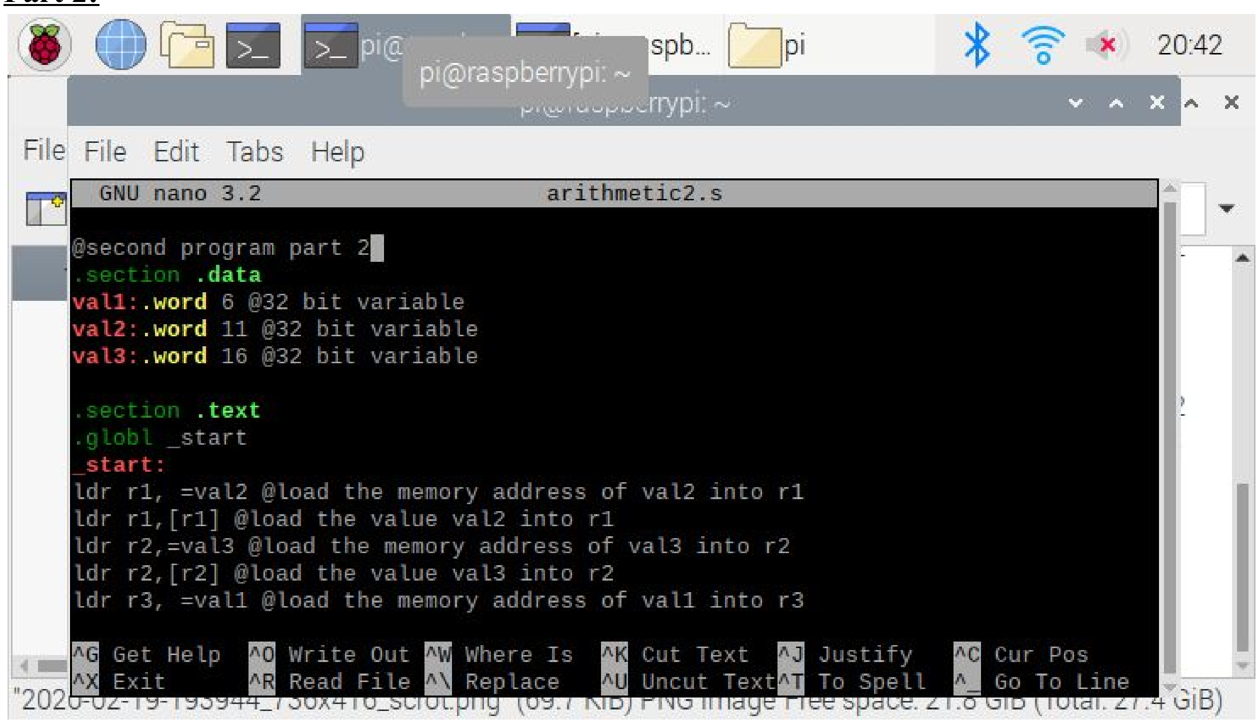


In this screenshot, I ran the program and then stepped into the next line and then displayed 3 of the memory addresses.

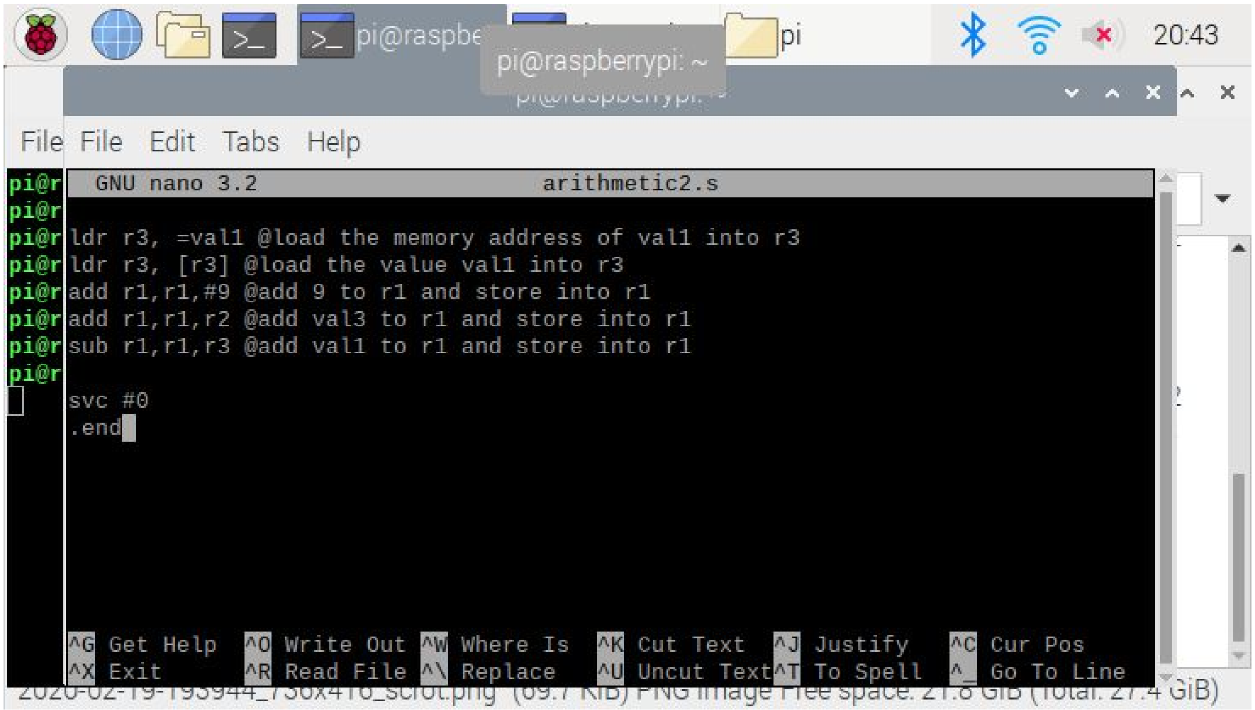


Here is a screenshot of the registers with the loaded values.

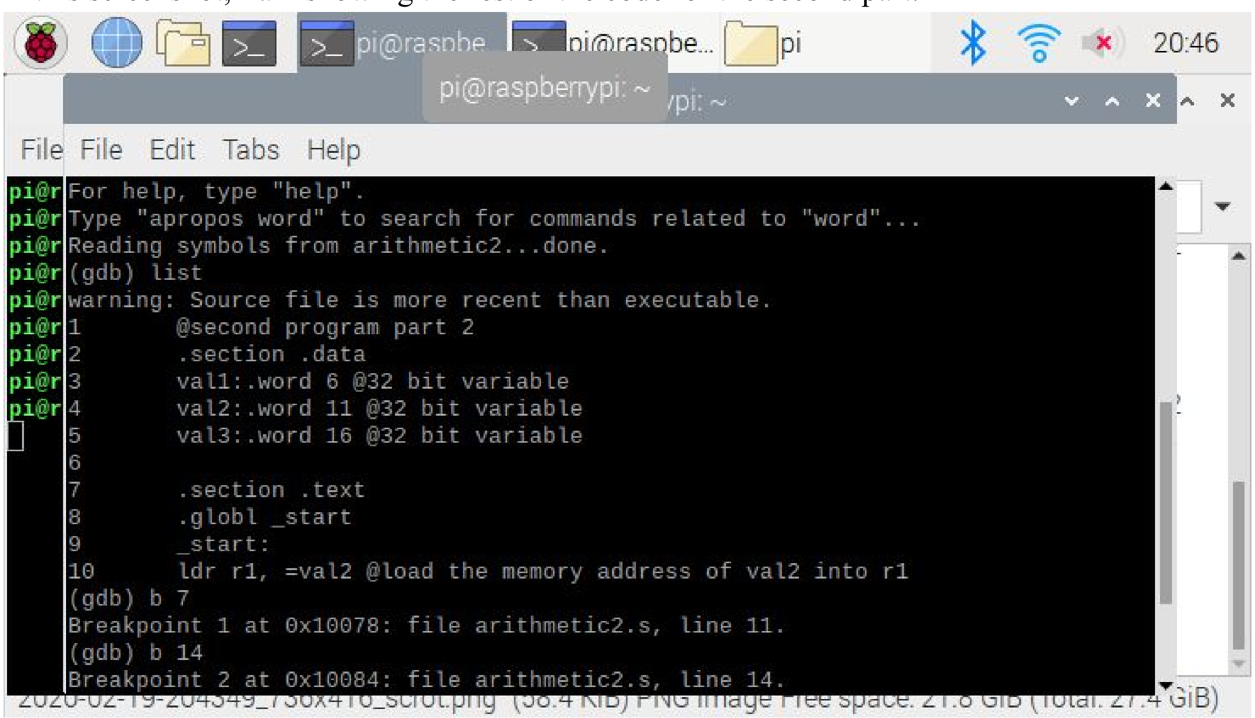
Part 2:



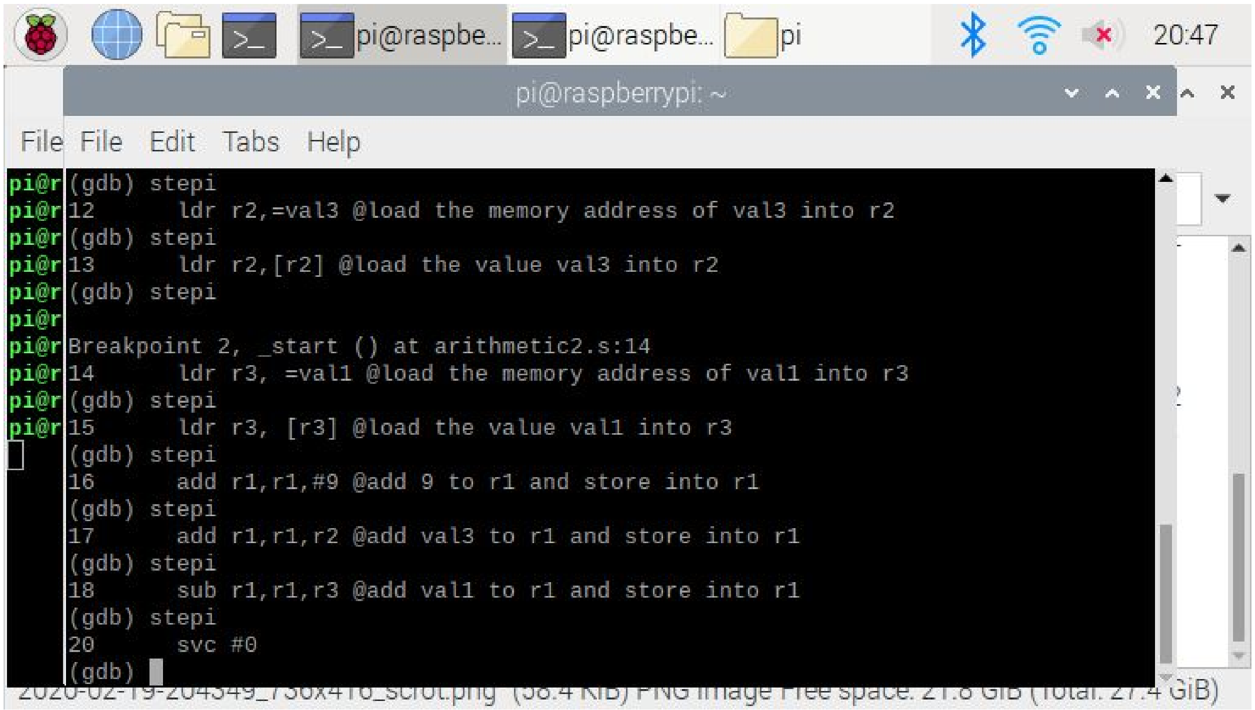
In this screenshot, I am displaying the code for the second part. In this program, I am loading the memory addresses into registers to which I then use the registers to perform some basic arithmetic.



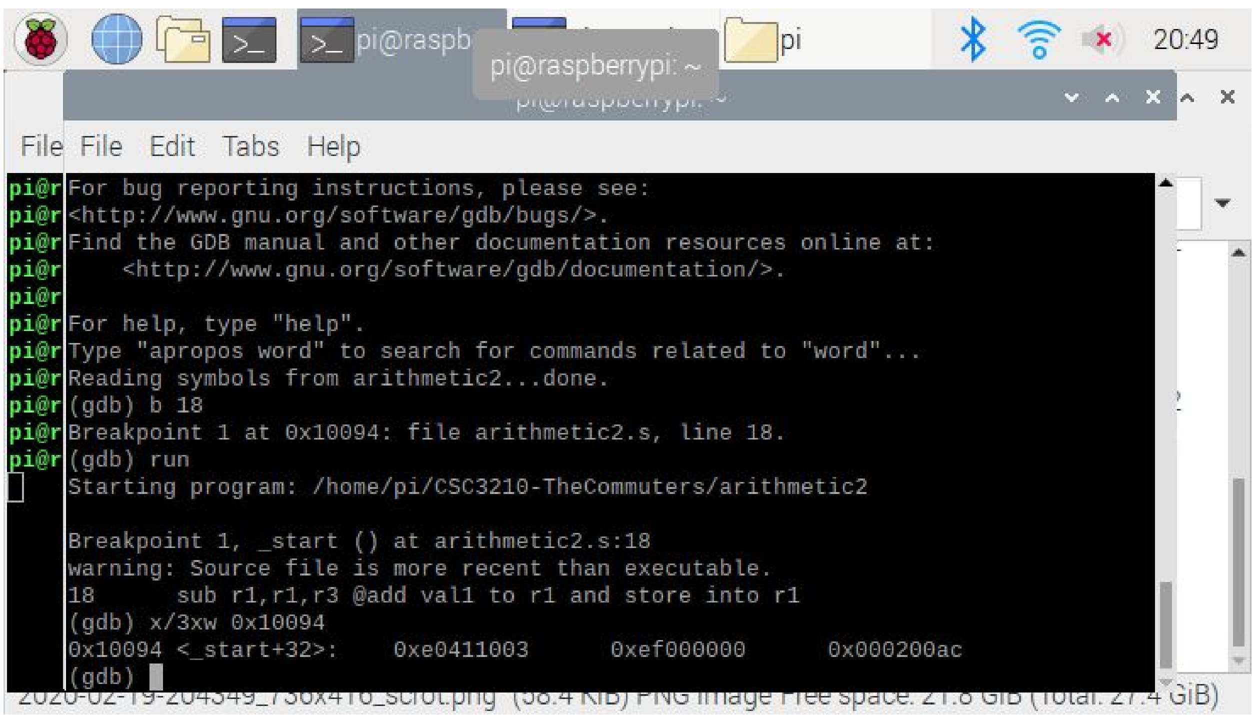
In this screenshot, I am showing the rest of the code for the second part.



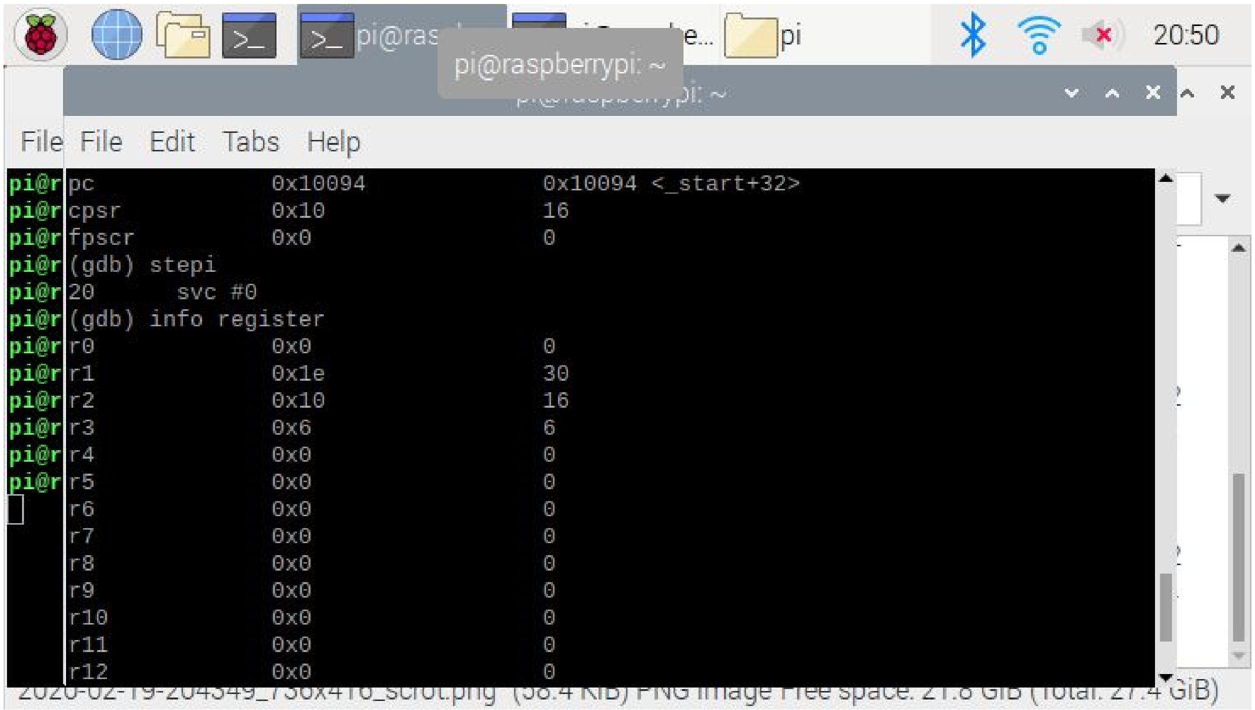
In this screenshot, I listed the first ten lines of my program and then placed a breakpoint at 7 and 14.



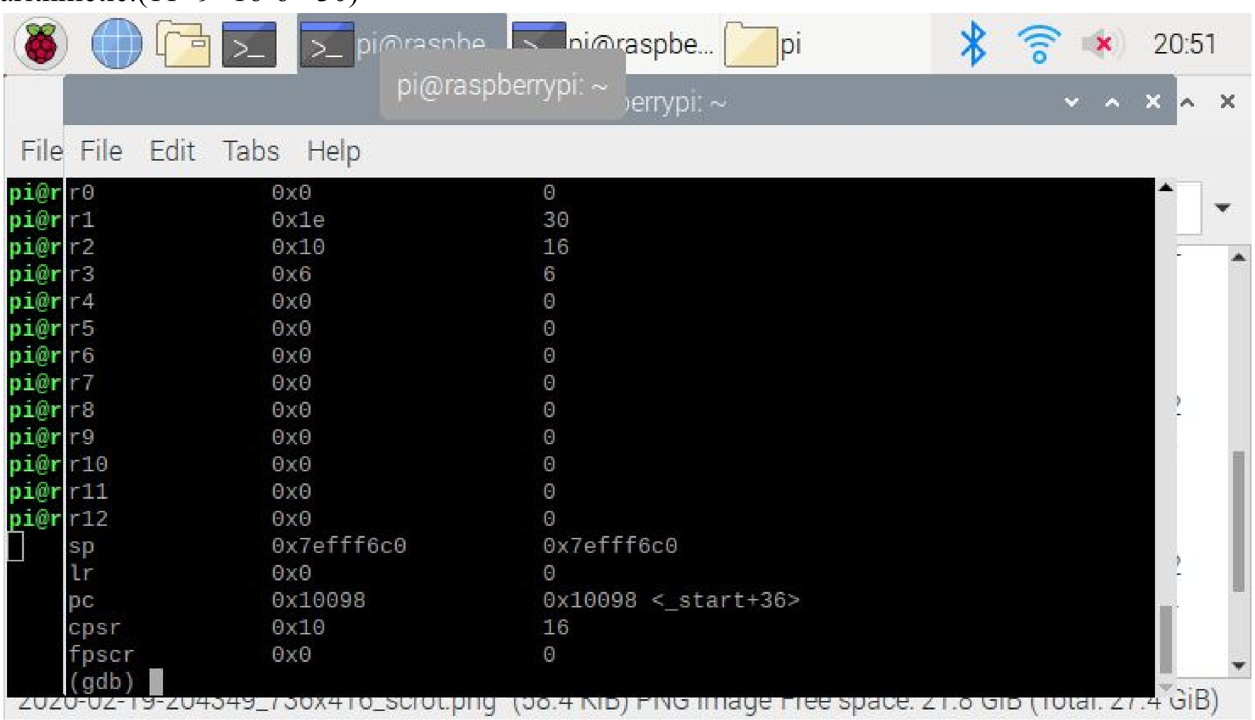
In this screenshot, I was stepping into each line of code to see what I would see, and it showed each line.



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 three 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.(11+9+16-6 =30)



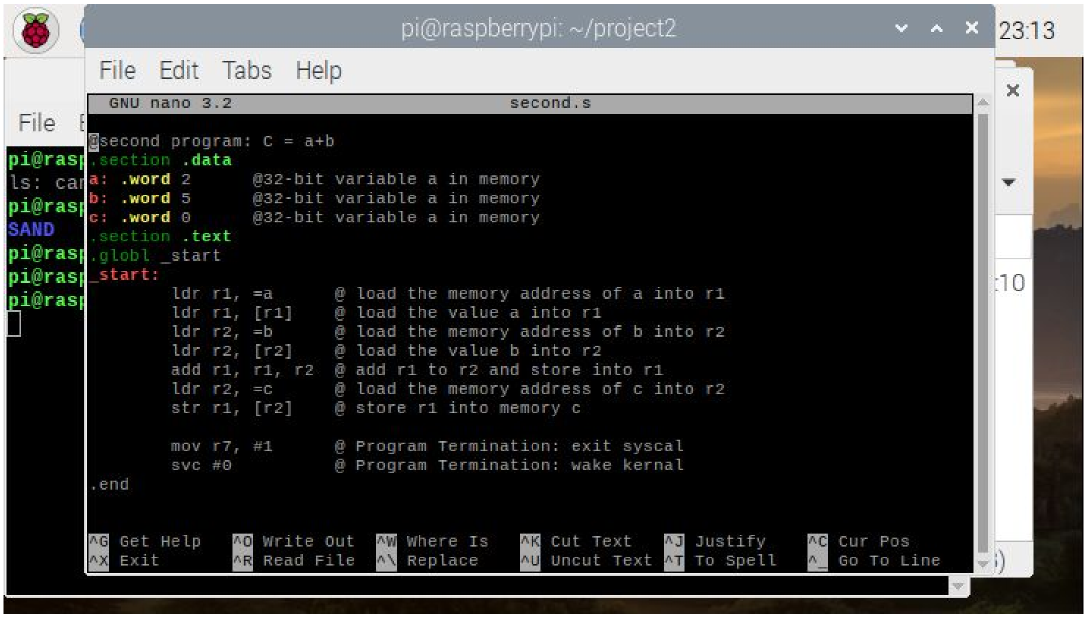
This is a screenshot of the rest of the registers.

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

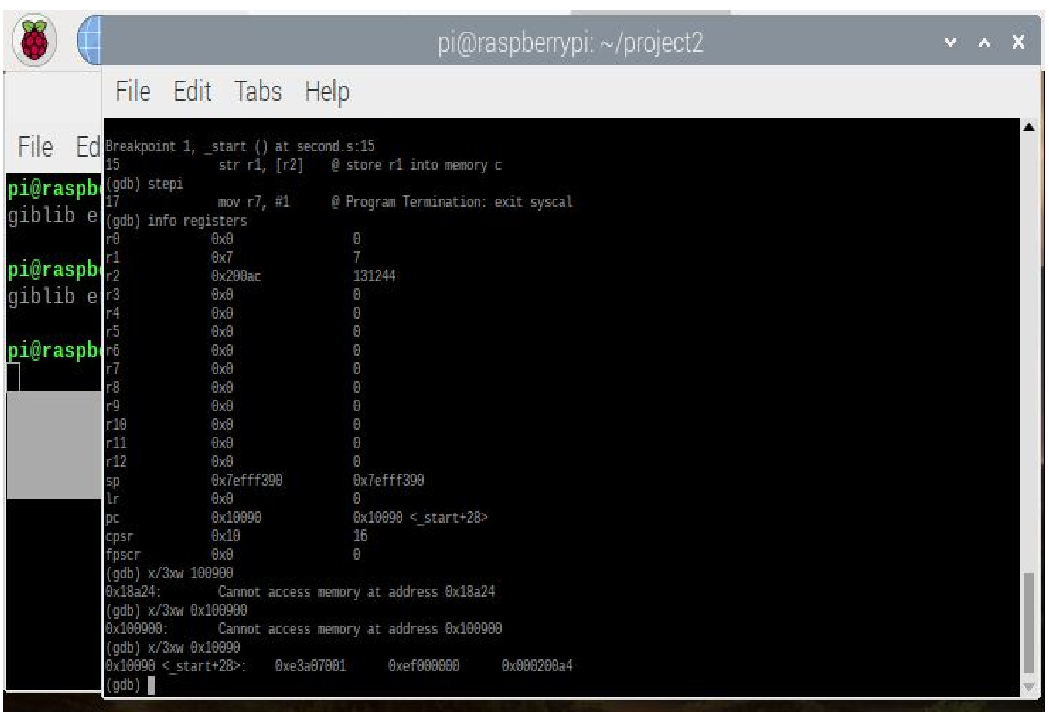
- **Identifying the components of the raspberry PI B+.**
  - The Raspberry Pi's components are a single board computer, quad -core multicore CPU, 1 GB RAM, 2 Usb ports, ethernet/ controller, Power port, Camera, HDMi, Power, and Display.
- **How many cores does the Raspberry Pi's B+ CPU have?**
  - .The 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

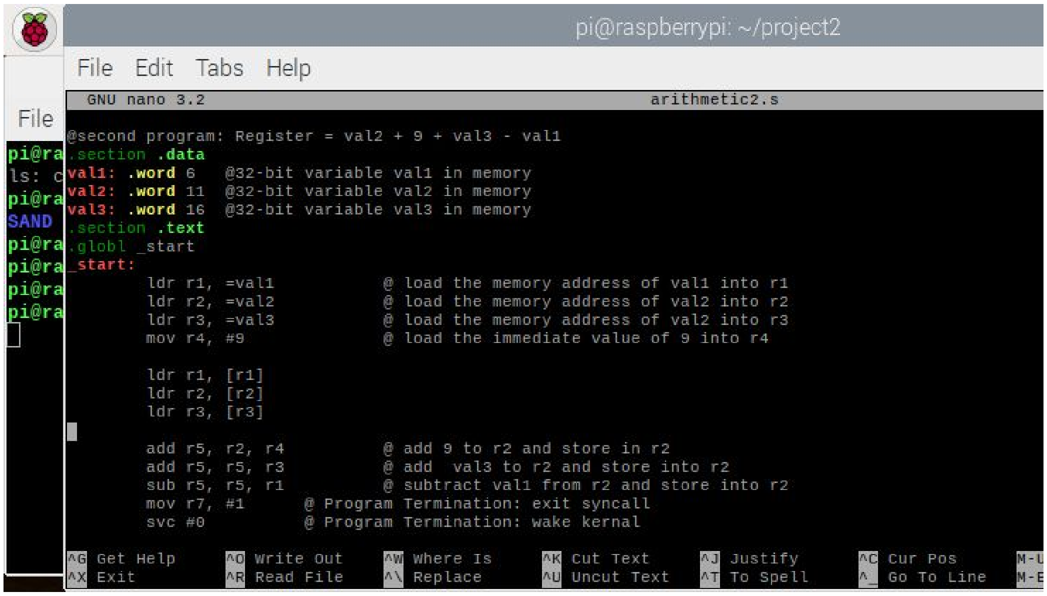
Parallel Programming Basics: Joan Galicia



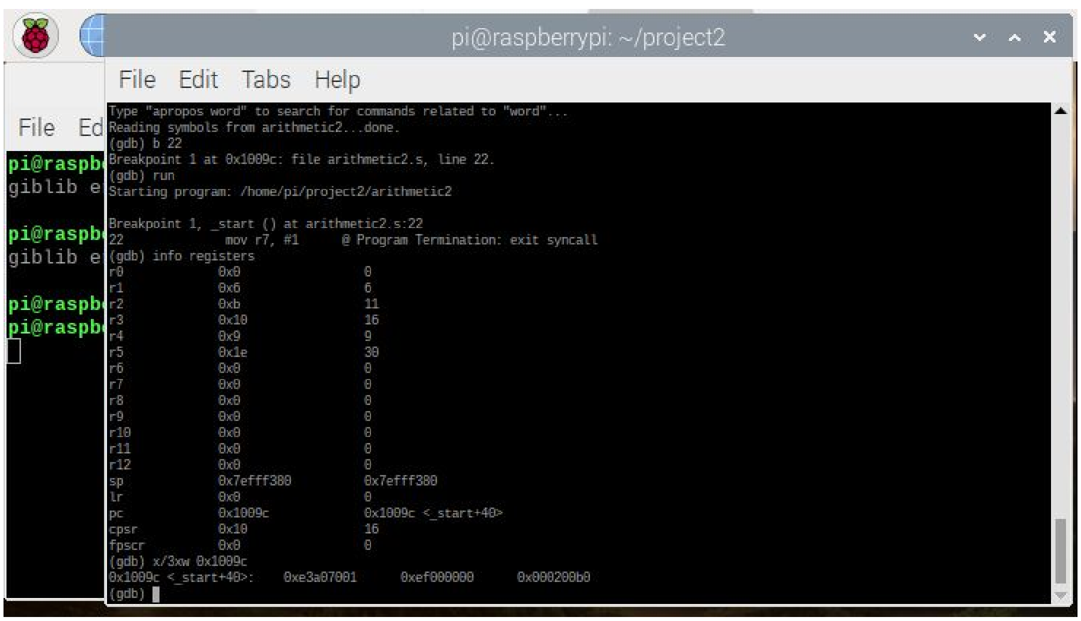
In this screenshot is the code written for second.s in the next screen shot is where the code is debugged. The only difference in this program is the way variables are used and called for. This language uses first assigns the memory address to the register and then uses load instead of move to assign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



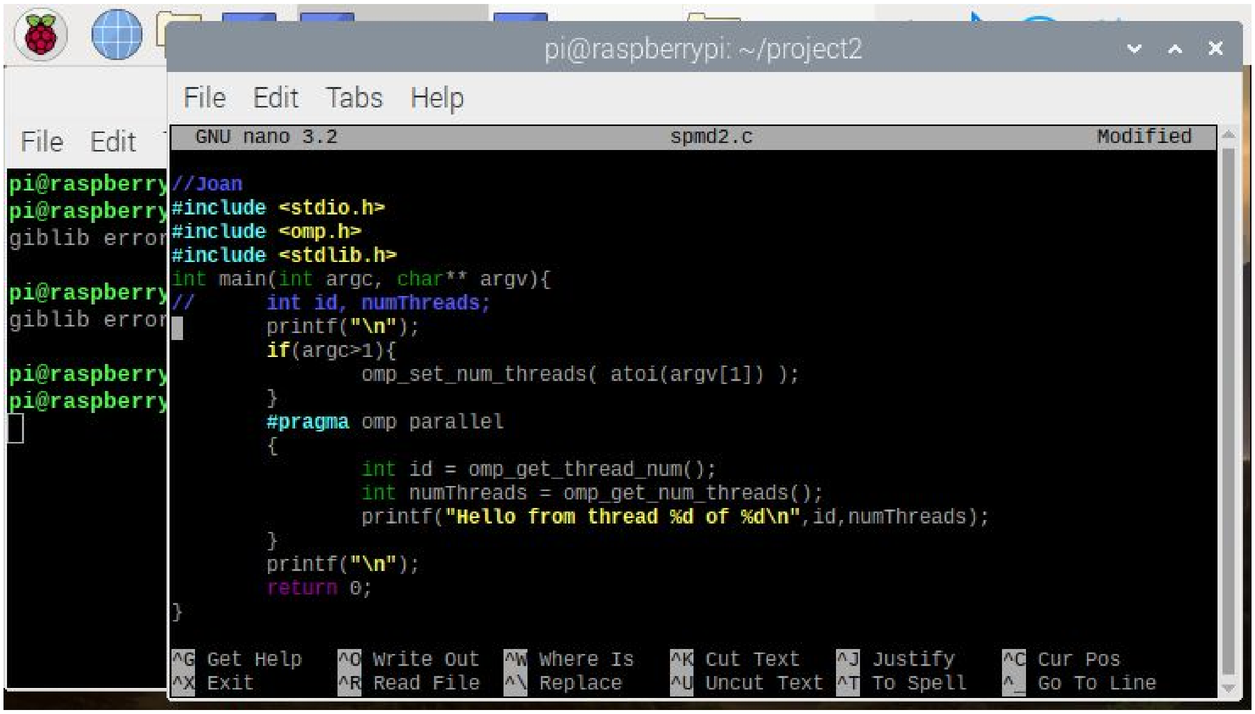
At breakpoint 15 the memory address resulted in 0x01008 and once “stepi” was commanded it then proceeded to the next line where the code is. This screenshot is set up at line 17 and so the memory registers gave this address 0x10090. After accessing this address it gave 3 more addresses that when trying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



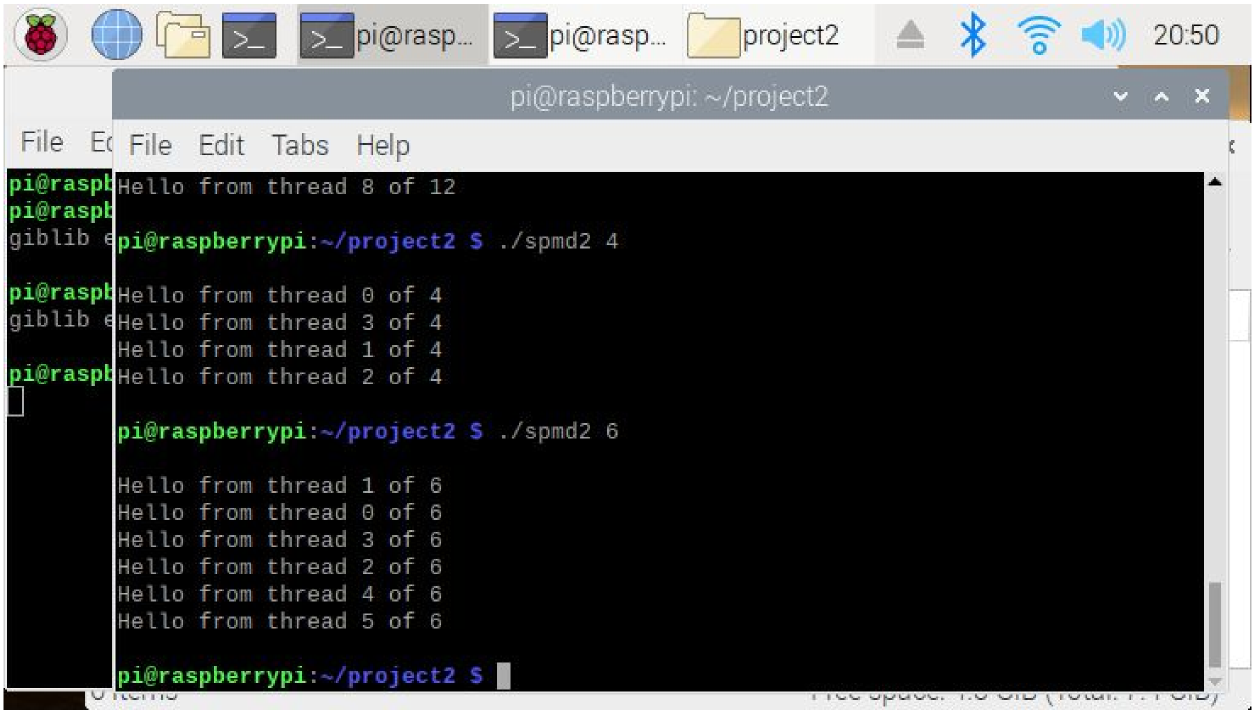
This screenshot is the code used for arithmetic2. From the past program, second, I learned to use ldr to load the memory addresses of my variables to the registers. This program was asking to add three values and subtract one value.



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 when I tried to compile the program into an executable as there were only simple errors being indicated.



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. The displays n number of lines based on the number given when executing the program.

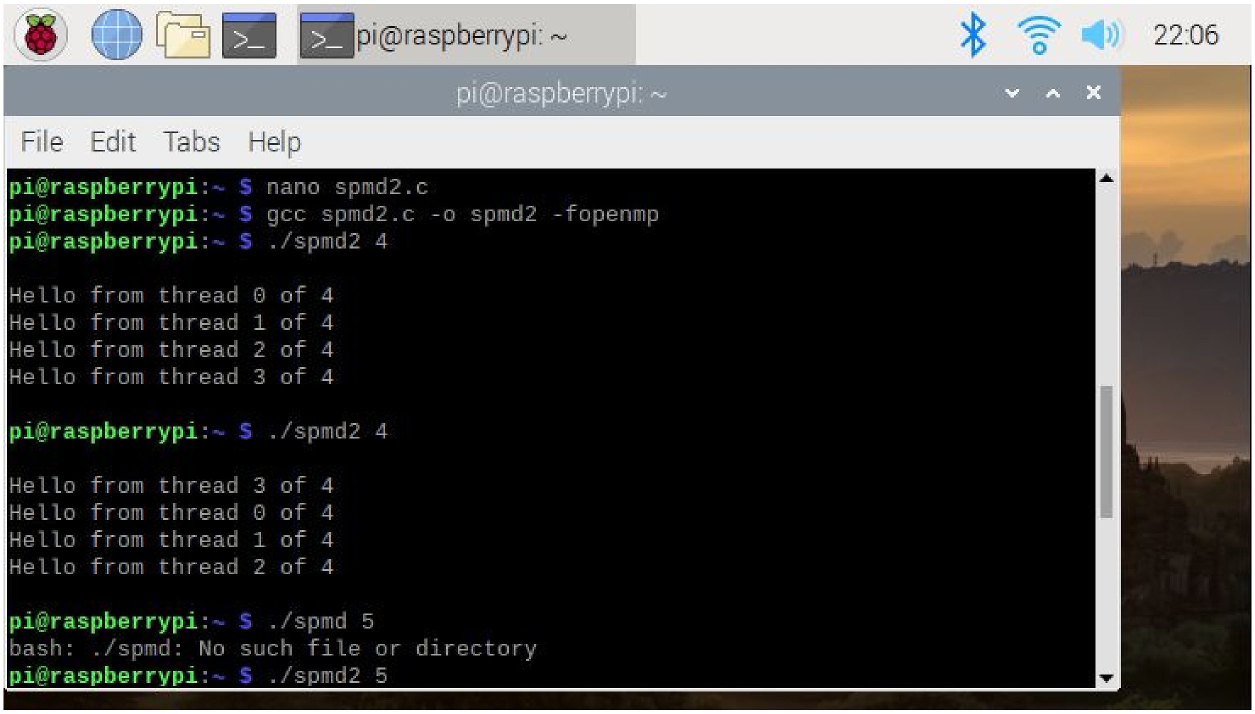
<u>Parallel Programming Skills Foundation: Andre Nguyenphuc</u>

- **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, Quad-core CPU
- **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 has more features in the instruction 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. Sequential computation costs less than parallel computation due to it only being done on a single processor, but will take longer than parallel computation.
  - 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. Parallel computation is faster than sequential computation and load on the processors is not high because one processor is not doing the whole task.
- **Identify the basic form of data and task parallelism in computational problems**
  - Data parallelism is dividing up data and doing the same work on different processors. An example that represents data parallelism is an assembly line because there are multiple people doing the same tasks.
  - Task parallelism is dividing up tasks to different processors. An example of task parallelism would be any group project where team members will have tasks divided to them to complete on their own.
- **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,

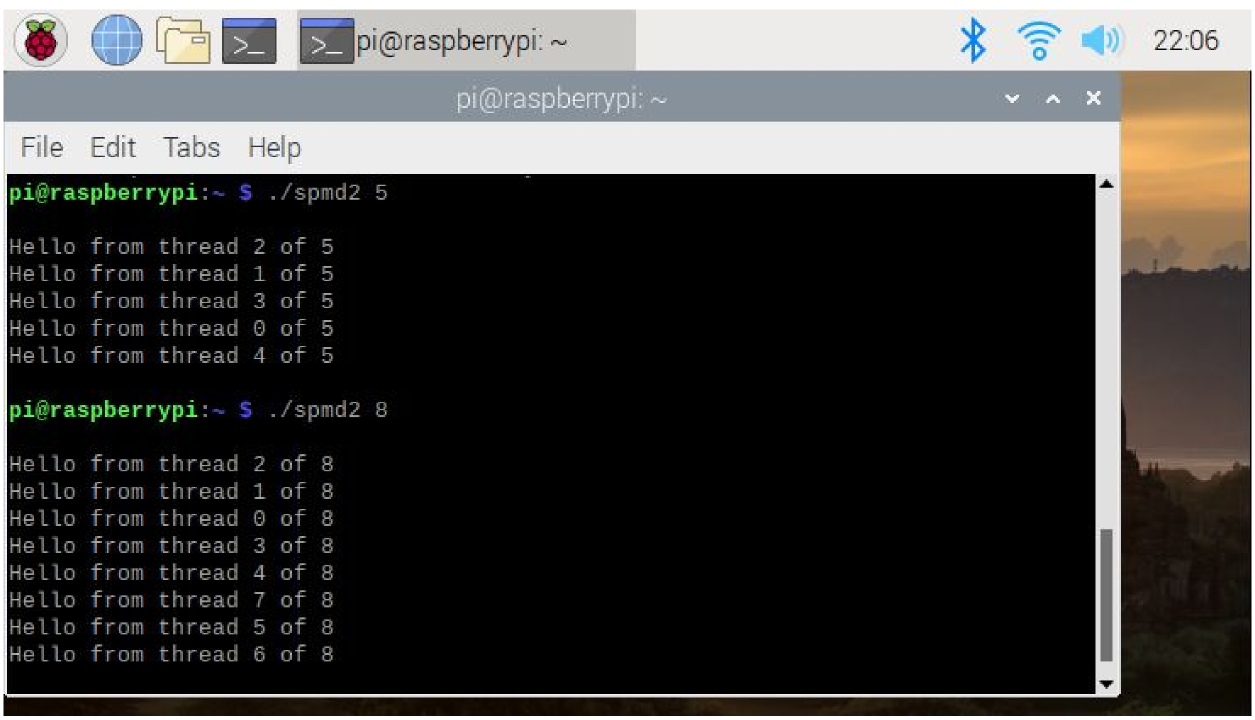
independent parts. A difference includes processes do not share memory with each other while threads all share a common memory of the process they belong to.

- **What is OpenMP and what is OpenMP pragmas?**
  - OpenMP is a library for parallel programming and all the threads share memory and data
  - 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 have faster execution than single-core due to parts of problems being solved concurrently
  - Uses less power than single-core

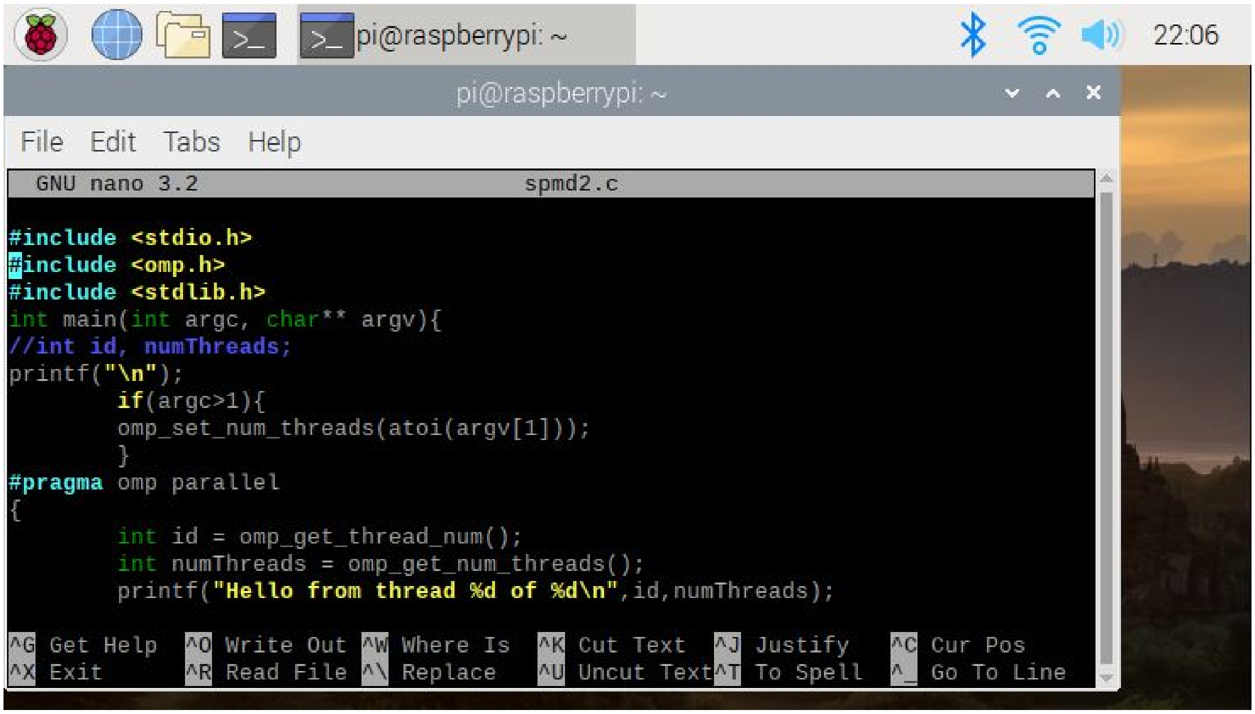
Parallel Programming Basics: Andre Nguyenphuc



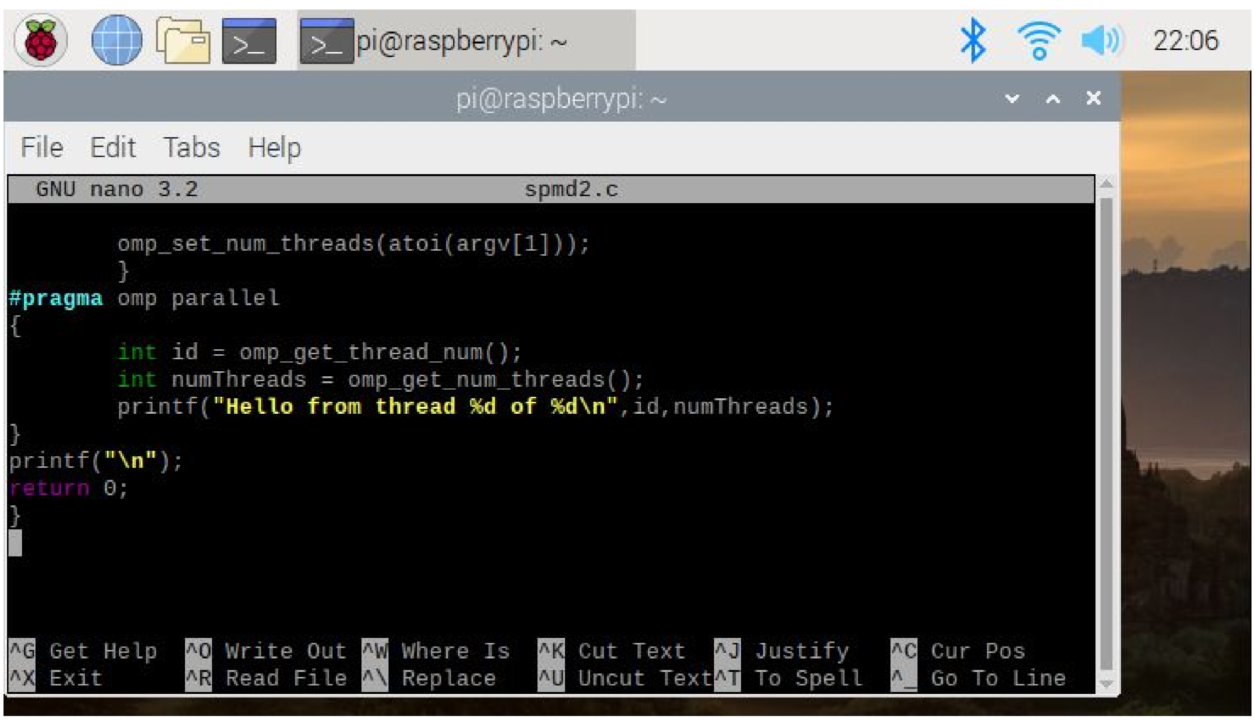
In this screenshot I compiled the spmd2 program then tested it with 4 which means how many threads to fork. This showed me that the order of the threads is out of order.



I tested more threads here to see if I can go past 4 threads and the program allowed me and like the 4 threads the order for the threads seem to be out of order.

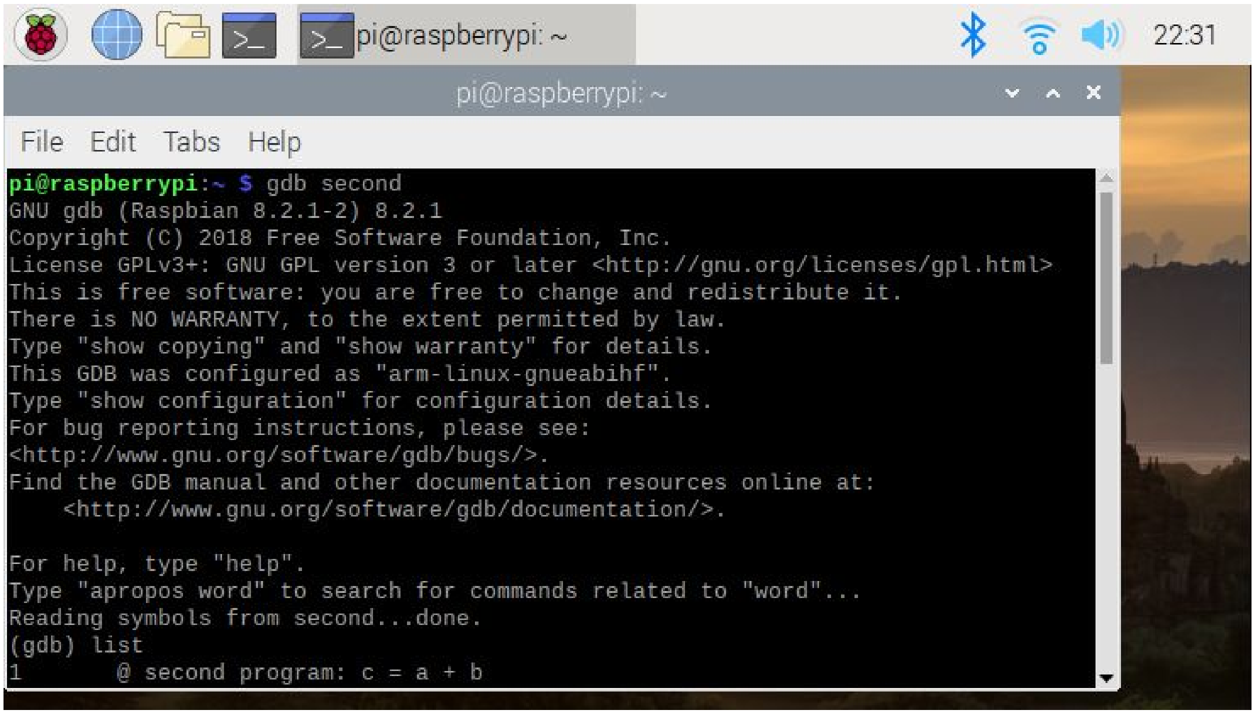


This is the C program which was something new for me. I observed that some commands were different from Java but mean the same thing like printf in java is system.out.print. I also had to make line 5 a comment and fix lines 12 and 13 to have id and numThreads be declared as variables.

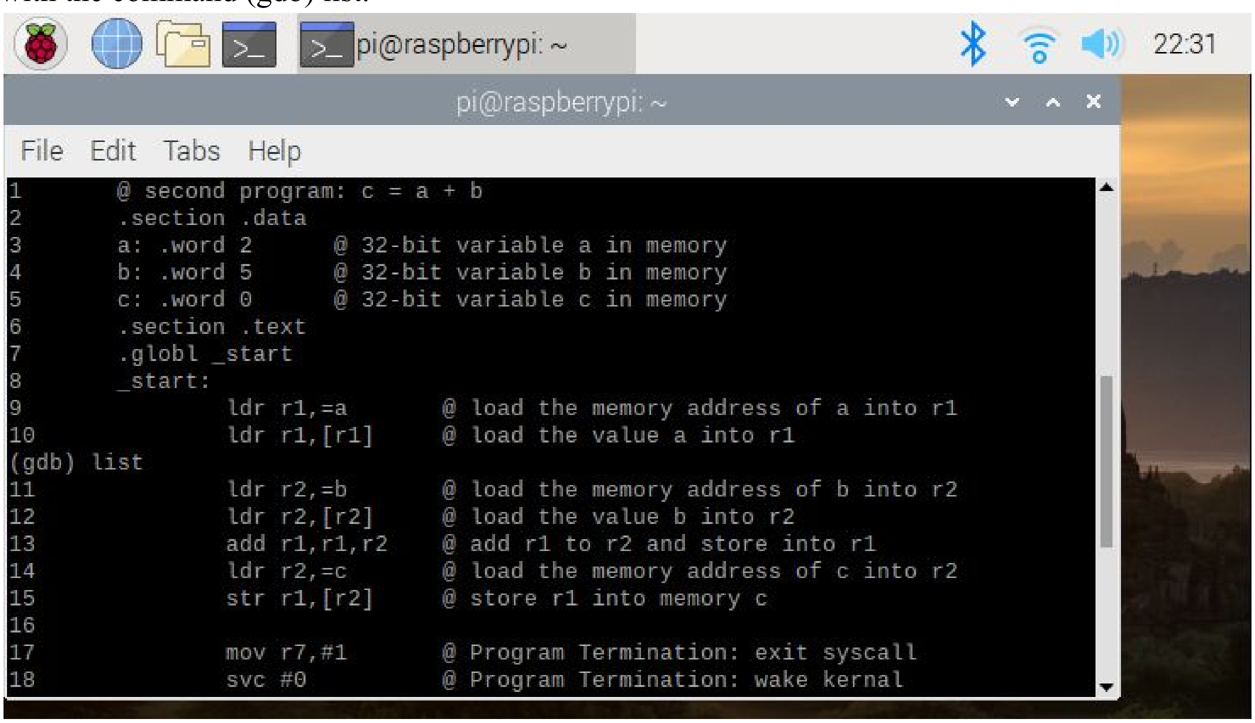


This is just the rest of the spmd2 code

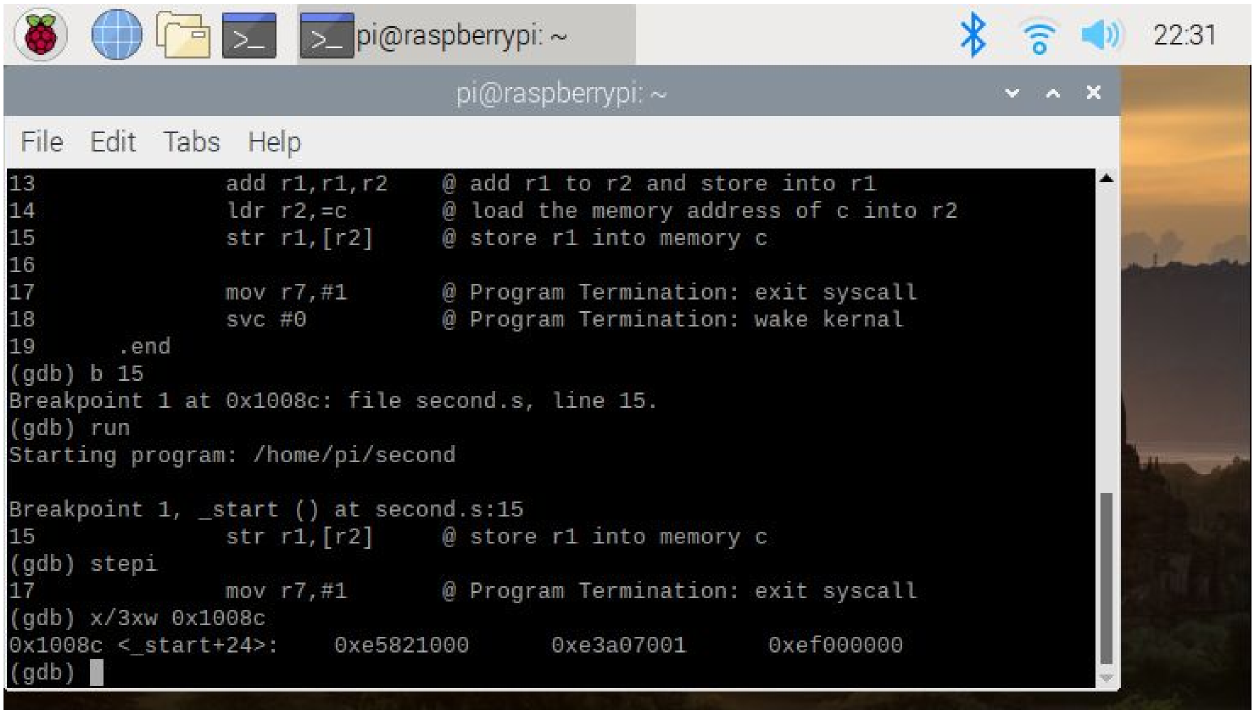
ARM Assembly Programming: Andre Nguyenphuc:



Here I am compiling the second program and executing it with gdb then trying to show the code with the command (gdb) list.

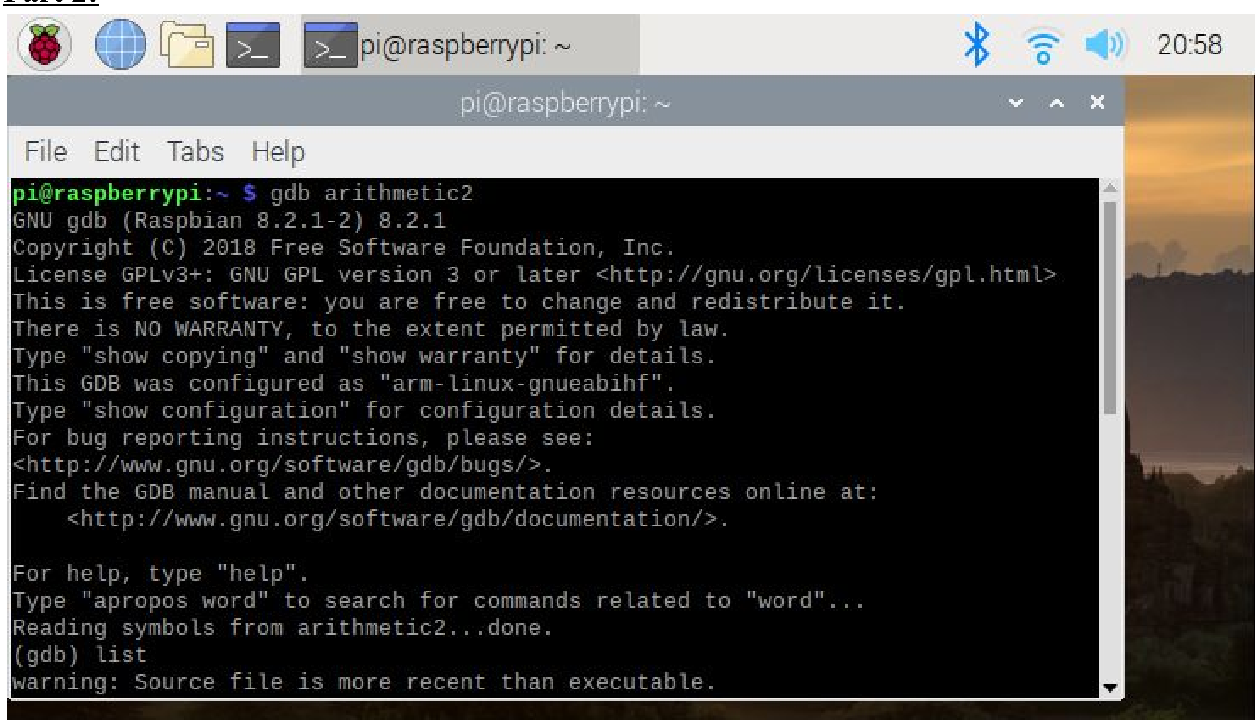


This is the code for the program and what I noticed is instead of moving like in the last assignment's program we store variables in the .data then we load the memory address and the value into the register we want.

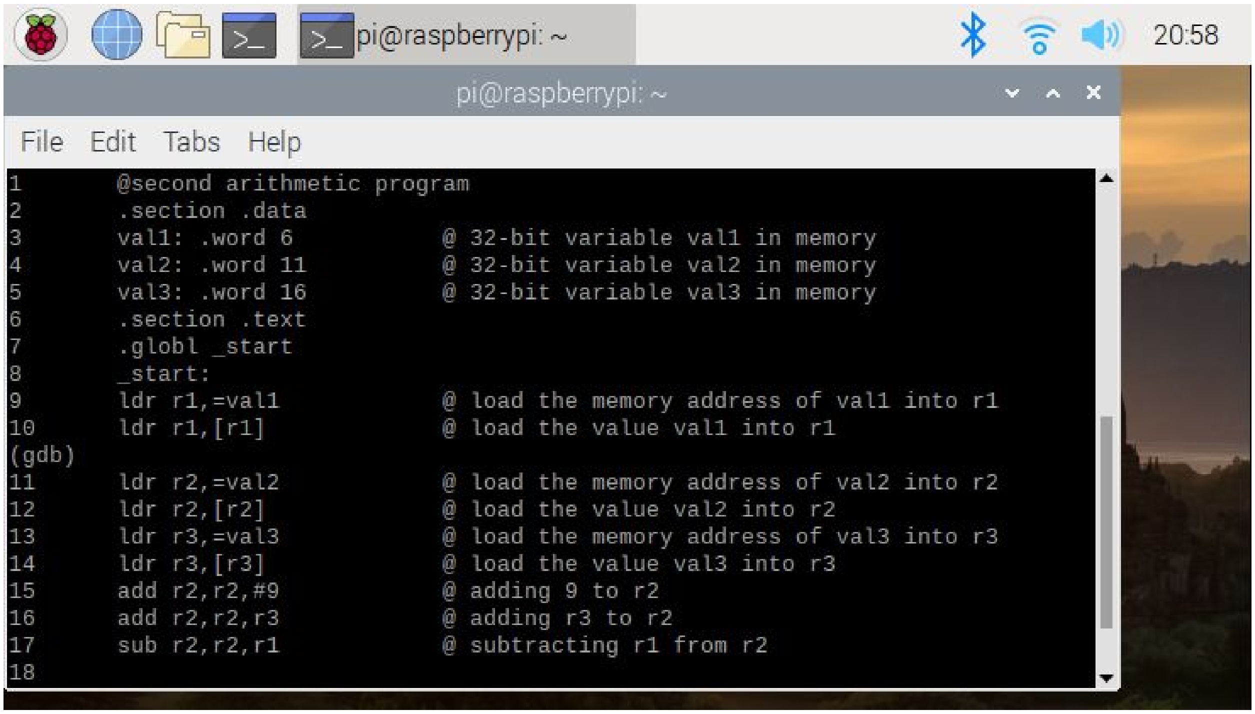


Here I set the breakpoint at 15 so the code could run then I tested the stepi which allowed me to continue even though I set a breakpoint and I displayed the memory which shows 0xe5821000, 0xe3a07001, and 0xef000000.

Part 2:

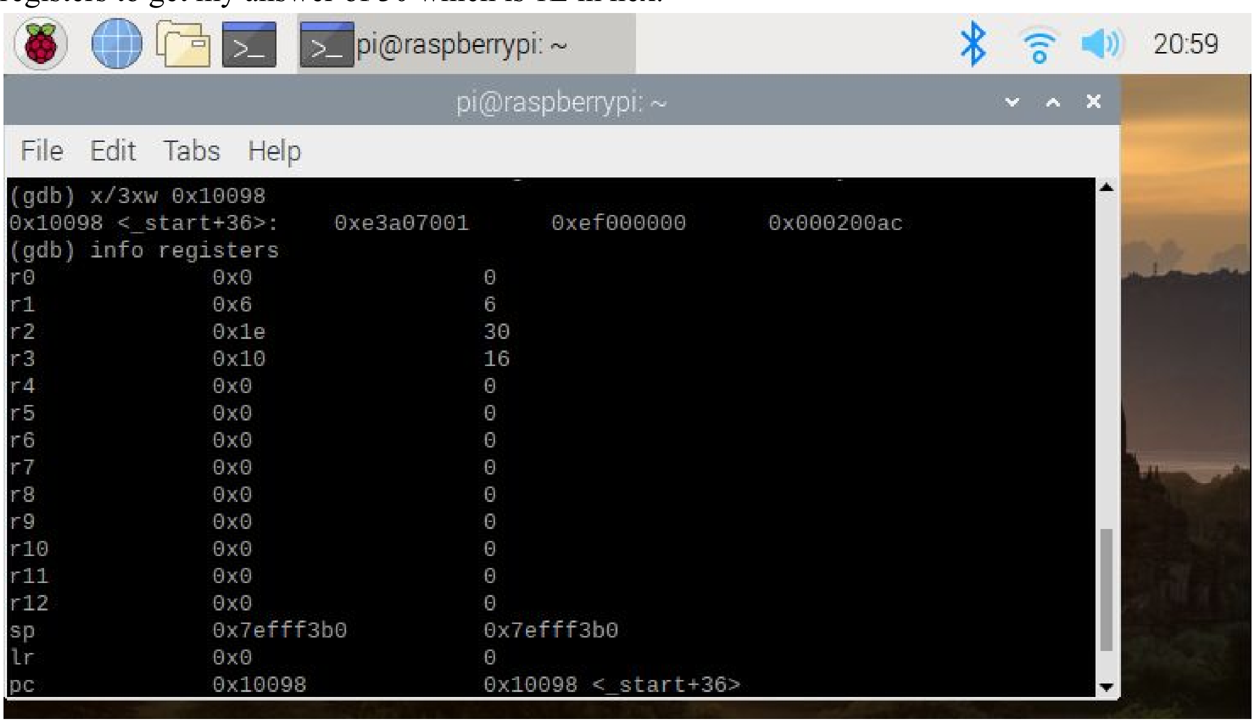


Here I am compiling and executing the arithmetic2 program




```
        @second arithmetic program
        .section .data
        val1: .word 6           @ 32-bit variable val1 in memory
        val2: .word 11          @ 32-bit variable val2 in memory
        val3: .word 16          @ 32-bit variable val3 in memory
        .section .text
        .globl _start
        _start:
        ldr r1,=val1            @ load the memory address of val1 into r1
        ldr r1,[r1]             @ load the value val1 into r1
(gdb)
        ldr r2,=val2            @ load the memory address of val2 into r2
        ldr r2,[r2]             @ load the value val2 into r2
        ldr r3,=val3            @ load the memory address of val3 into r3
        ldr r3,[r3]             @ load the value val3 into r3
        add r2,r2,#9            @ adding 9 to r2
        add r2,r2,r3            @ adding r3 to r2
        sub r2,r2,r1            @ subtracting r1 from r2
```

Here is my code which I first stored the immediate values inside variables in the .data. I then loaded different registers with the memory address and value of a certain variable. For example I loaded register 1 with val1's memory address and value of 6. I then did arithmetic with the registers to get my answer of 30 which is 1E in hex.




```
(gdb) x/3xw 0x10098
0x10098 <_start+36>:    0xe3a07001      0xef000000      0x000200ac
(gdb) info registers
r0             0x0                 0
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             0x10098             0x10098 <_start+36>
```

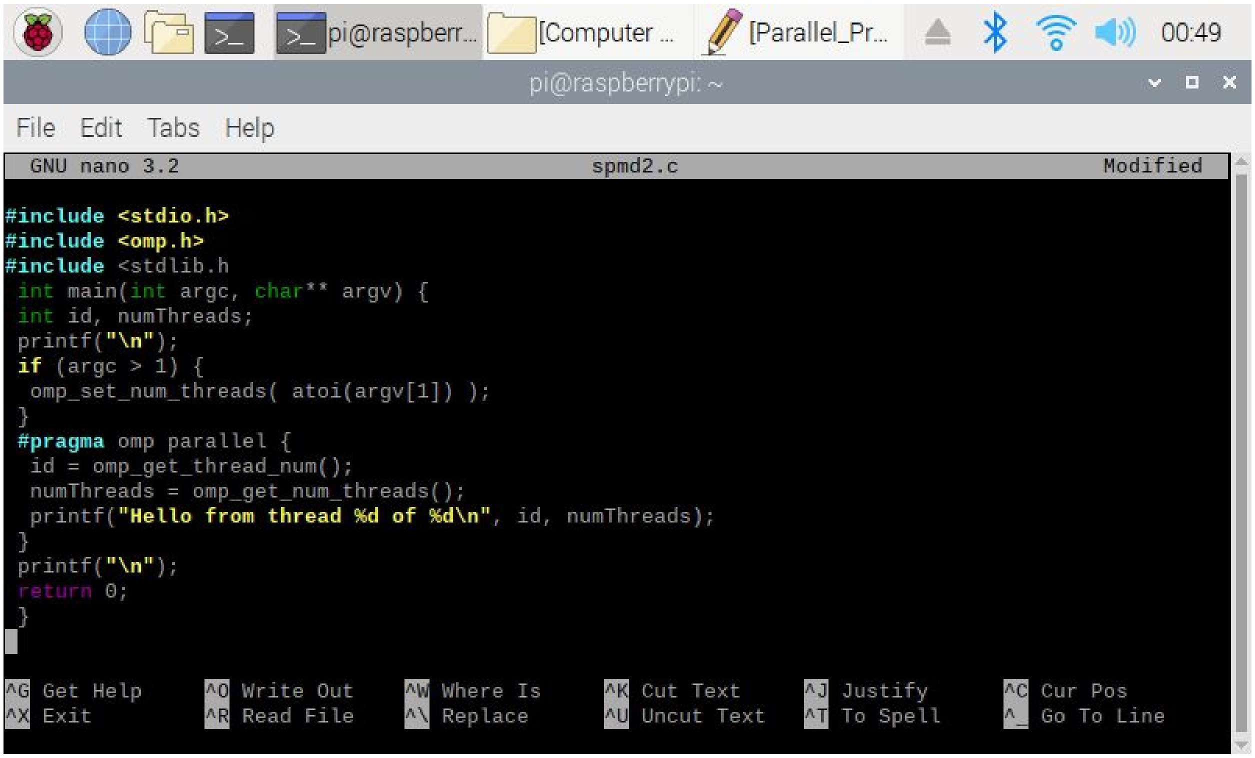
Here I am displaying the memory and the info registers which show that in my register 2 I get my desired value which is 30 and 1E in hex. In register 1 it is 6 and in register 3 it is 16 because that was the initial value I loaded it with. It stayed the same because I only used them to add and sub to register 2.

Parallel Programming Skills Foundation: Miguel Romo

- **Identifying the components on the raspberry PI B+**
  - USB, ethernet, ethernet controller, camera, HDMI, power, display, and CPU/RAM.
- **How many cores does the Raspberry Pi's B+ CPU have**
  - The Raspberry Pi's B+ CPU has a quad-core multicore CPU.
- **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).**
  - CICS has more processing power and has an instruction set that can perform complex instructions to access memory. ARM has a simple instruction set that can be executed more quickly. CISC put less effort on the compiler and relies on more hardware to decode and execute instructions. ARM has less hardware and relies on the compiler to decode and execute instructions. ARM can execute instructions more quickly and CICS.
- **What is the difference between sequential and parallel computation and identify the practical significance of each?**
  - Sequential computation completes a task one by one, only uses one processor, and requires more time to complete a task. Parallel computation completes multiple tasks at a time, multiple processors are used, and requires less time to complete tasks.
- **Identify the basic form of data and task parallelism in computational problems.**
  - Data parallelism refers to a broad category of parallelism in which the same computation is applied to multiple data items. Task parallelism applies to solutions where parallelism is organized around the functions to be performed rather than around the data.
- **Explain the differences between processes and threads.**
  - Threads share access to the memory, so threads can communicate with other threads by reading from or writing to memory that is visible to them all. A process is a thread of control that has its own private address space.
- **What is OpenMP and what is OpenMP pragmas?**
  - OpenMP is a low-level thread package that allows shared memory multiprocessor programming on different operating systems. OpenMP programs is a compiler directive that enables the compiler to generate threaded codes.
- **What applications benefit from multi-core (list four)?**
  - Database servers
  - Web servers
  - Compilers
  - Multimedia applications
- **Why Multicore? (why not single core, list four)**
  - To process more tasks, single-core clock frequencies are too slow and deliver lower performance.
  - Higher performance

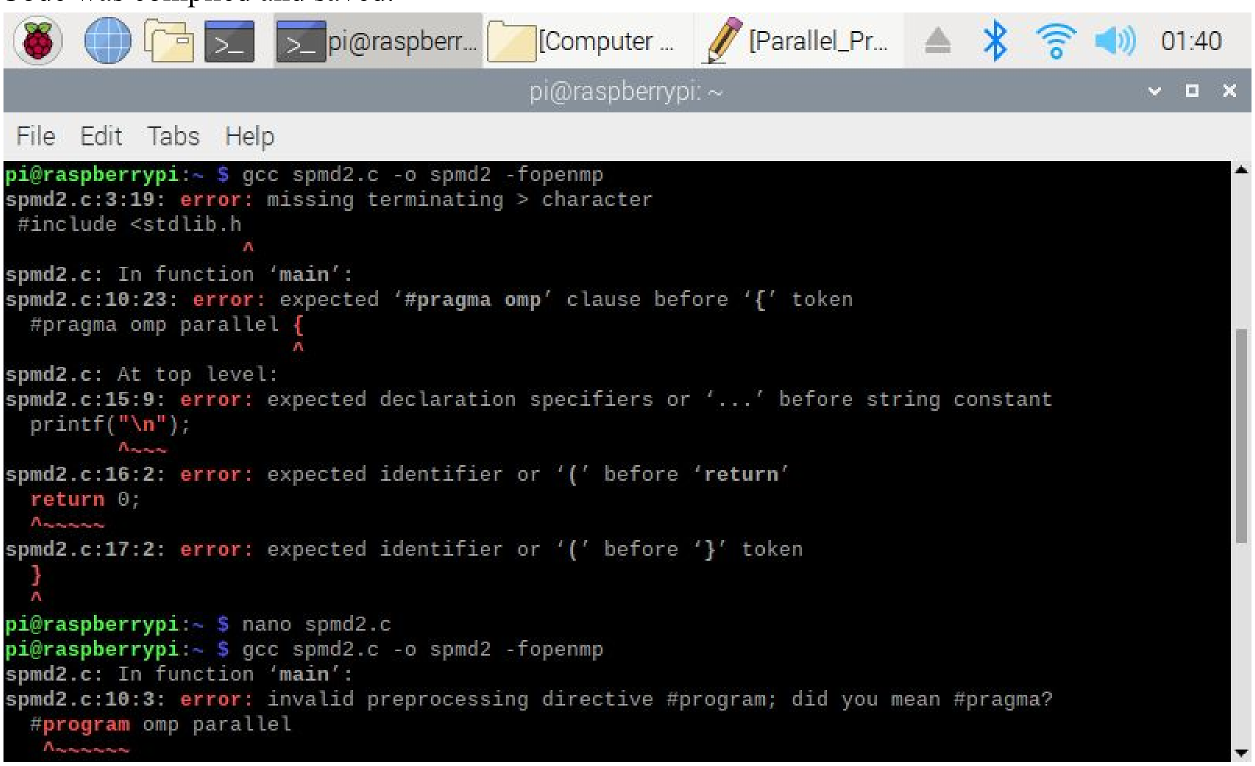
- Lower energy cost
- Lower power cost

Parallel Programming Basics: Miguel Romo



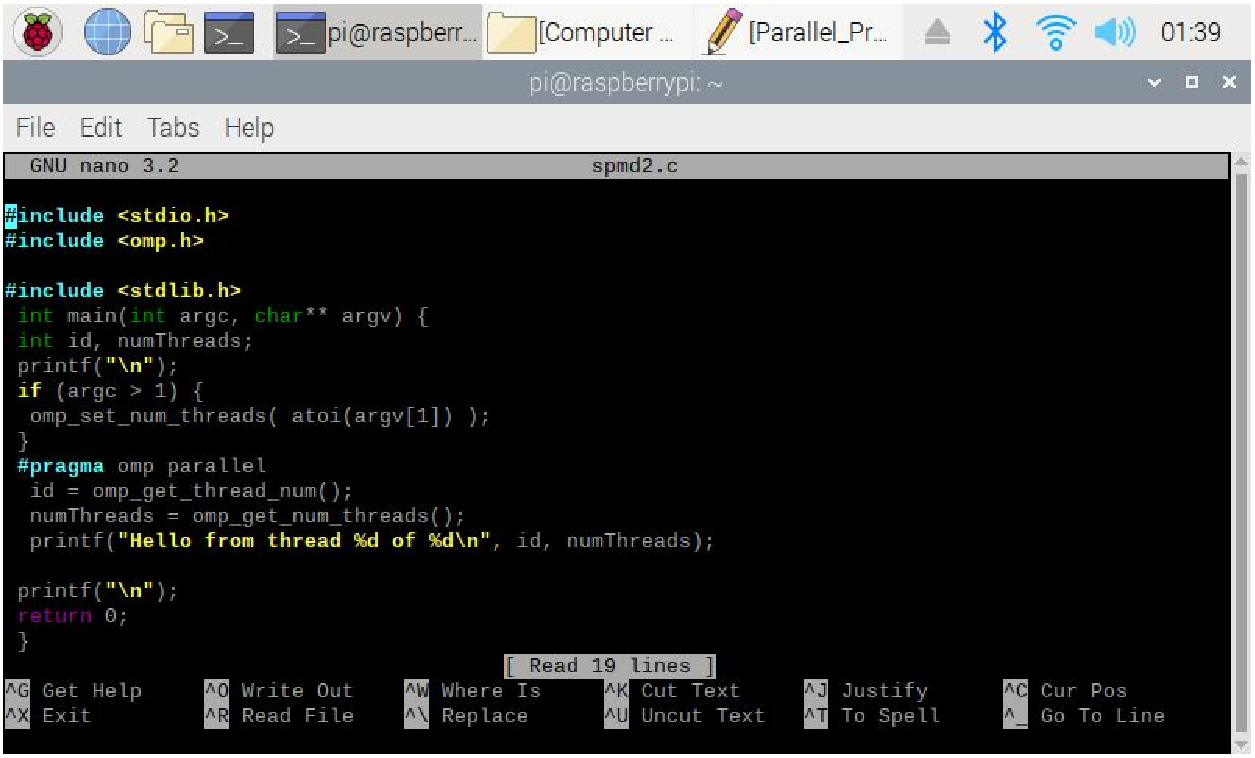
```
GNU nano 3.2                    spmd2.c                    Modified

#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 {
  id = omp_get_thread_num();
  numThreads = omp_get_num_threads();
  printf("Hello from thread %d of %d\n", id, numThreads);
 }
 printf("\n");
 return 0;
 }

^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
```

Code was compiled and saved.

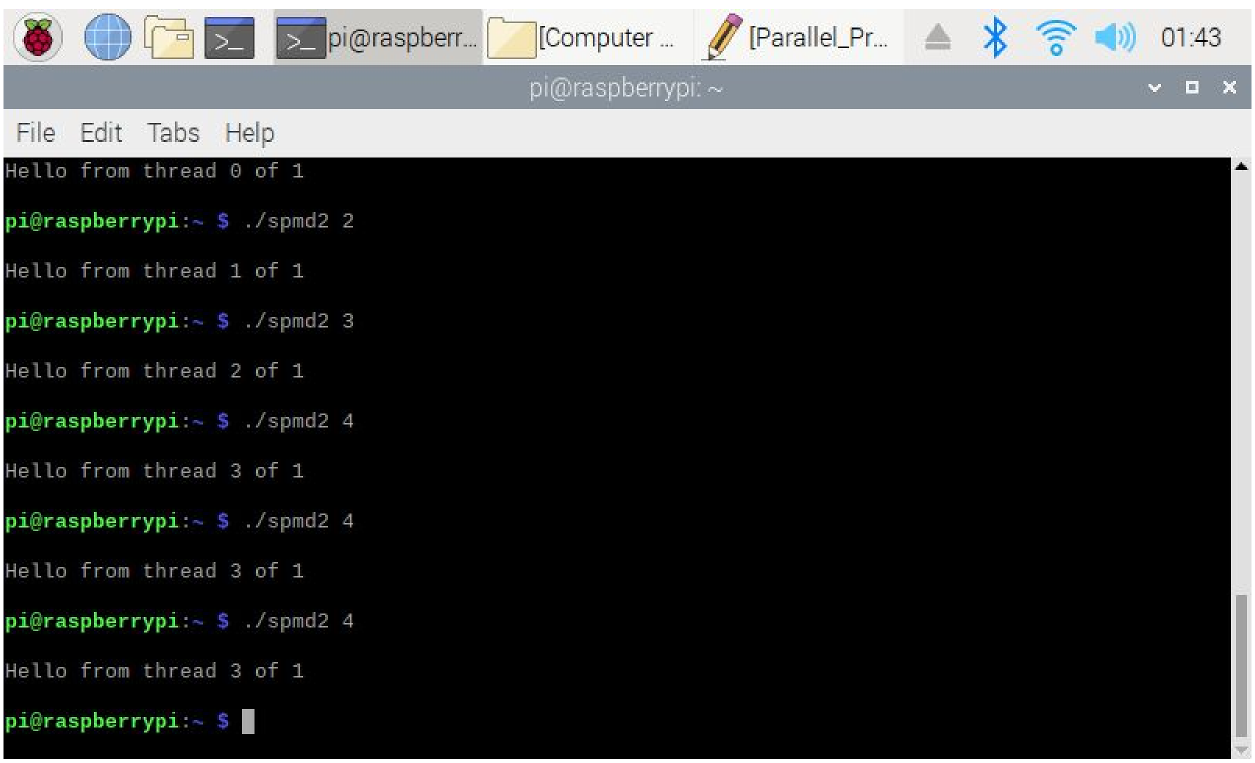


```
pi@raspberrypi:~ $ gcc spmd2.c -o spmd2 -fopenmp
spmd2.c:3:19: error: missing terminating > character
 #include <stdlib.h
                   ^
spmd2.c: In function 'main':
spmd2.c:10:23: error: expected '#pragma omp' clause before '{' token
  #pragma omp parallel {
                       ^
spmd2.c: At top level:
spmd2.c:15:9: error: expected declaration specifiers or '...' before string constant
  printf("\n");
         ^~~~
spmd2.c:16:2: error: expected identifier or '(' before 'return'
  return 0;
  ^~~~~~
spmd2.c:17:2: error: expected identifier or '(' before '}' token
  }
  ^
pi@raspberrypi:~ $ nano spmd2.c
pi@raspberrypi:~ $ gcc spmd2.c -o spmd2 -fopenmp
spmd2.c: In function 'main':
spmd2.c:10:3: error: invalid preprocessing directive #program; did you mean #pragma?
  #program omp parallel
   ^~~~~~~
```

Executable program was created, and ran, but errors were found.



Errors in the code were corrected.



Program ran successfully but the same thread id number appears more than once.

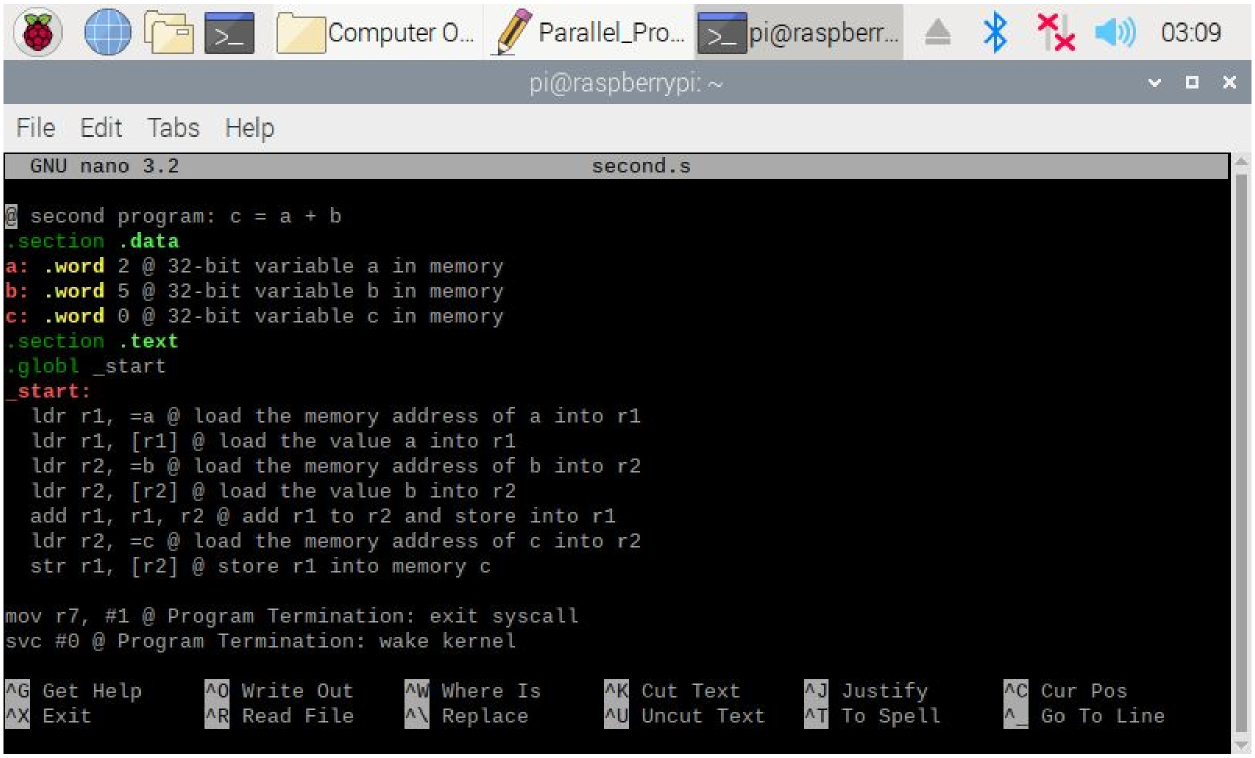
```
GNU nano 3.2                                spmd2.c

#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);
}
 printf("\n");
 return 0;
 }

^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 fix this we need to declare our variables inside the block where the program will be forked and run in parallel on separate threads.

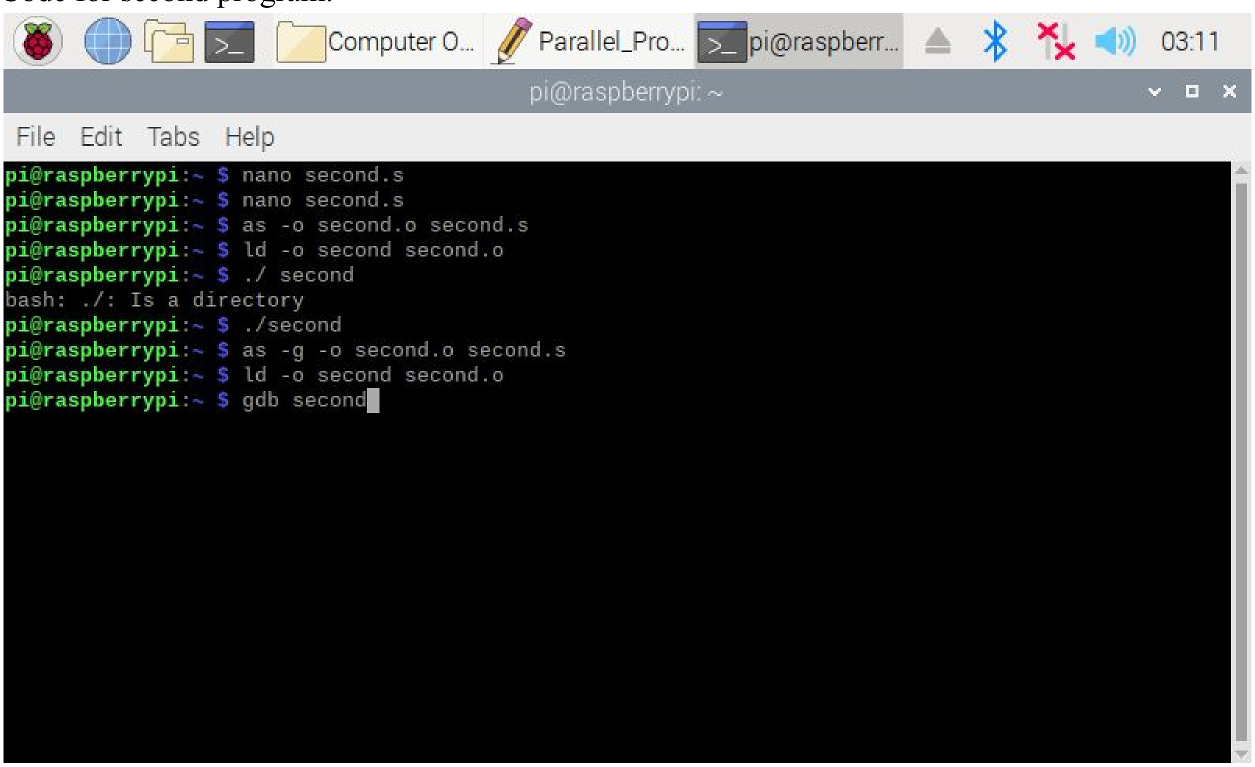
```
pi@raspberrypi:~ $ gcc spmd2.c -o spmd2 -fopenmp
pi@raspberrypi:~ $ ./spmd2 4

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:~ $ ./spmd2 4

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

pi@raspberrypi:~ $ ./spmd2 4

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

pi@raspberrypi:~ $
```

After declaring the variable in the right place, we can see that different threads IDs are now being used.

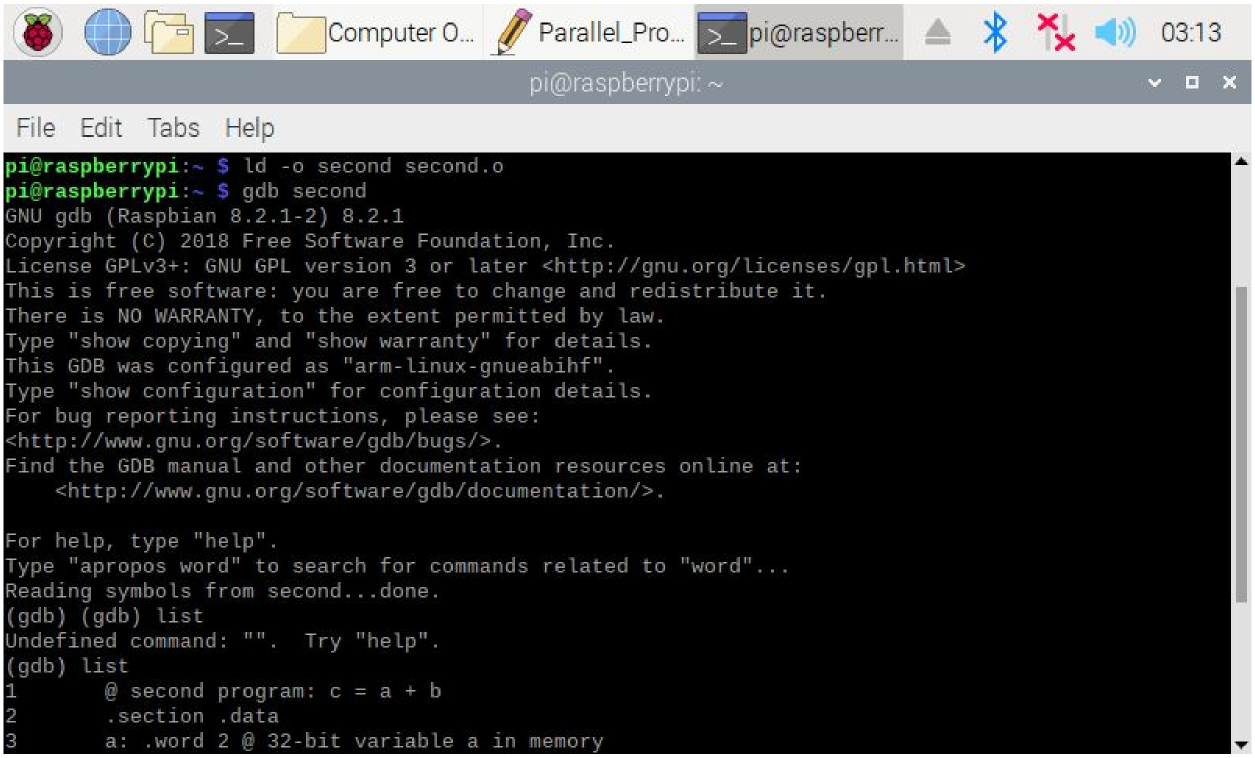
<u>ARM Assembly Programming: Miguel Romo</u>



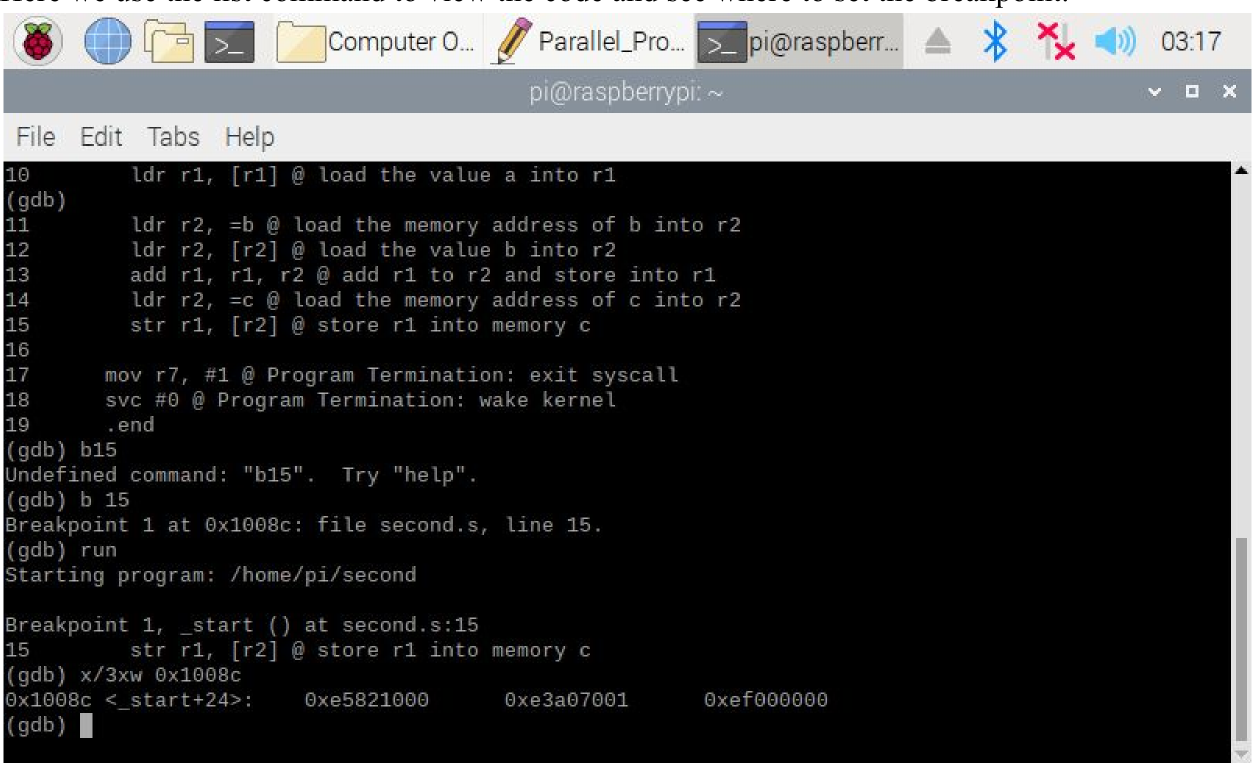
Code for second program.



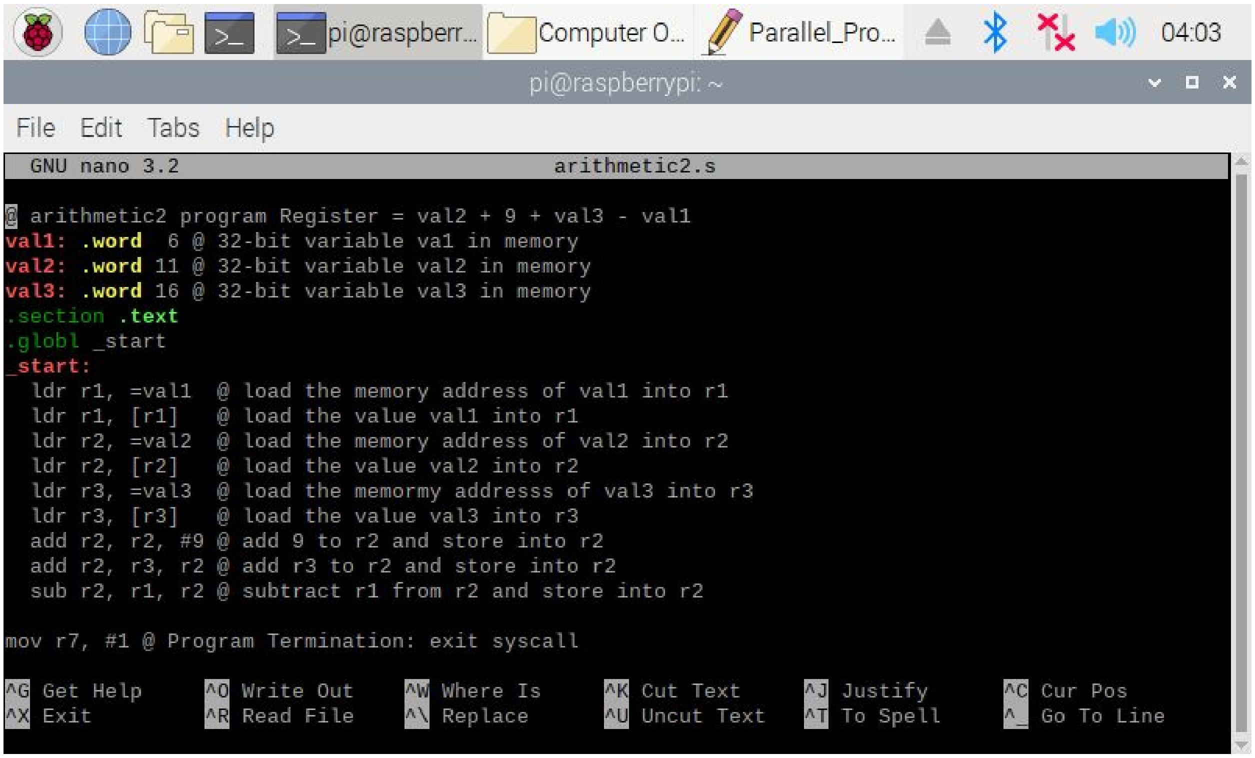
The commands used to create a file name for the program, and assemble link and run program. No output is shown in the terminal window, so we put the program in debug by using the GDB commands to examine the contents of the registers and memory.



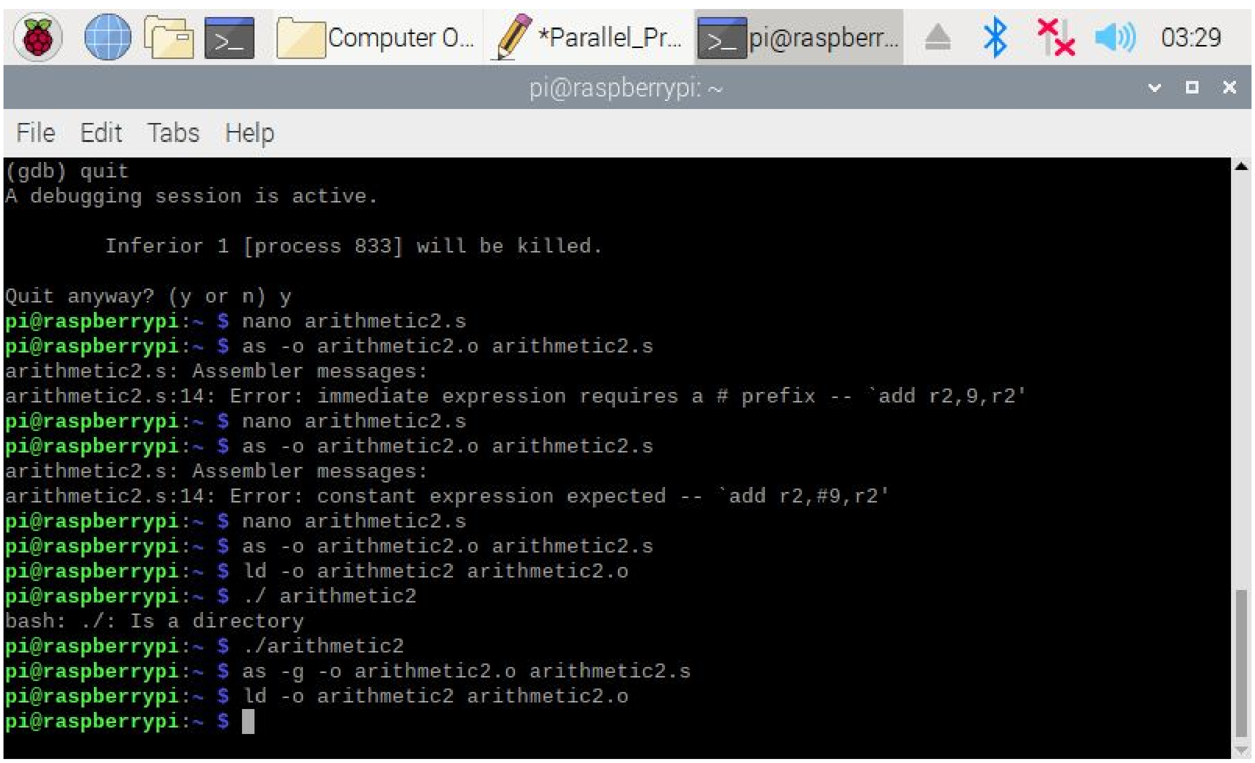
Here we use the list command to view the code and see where to set the breakpoint.



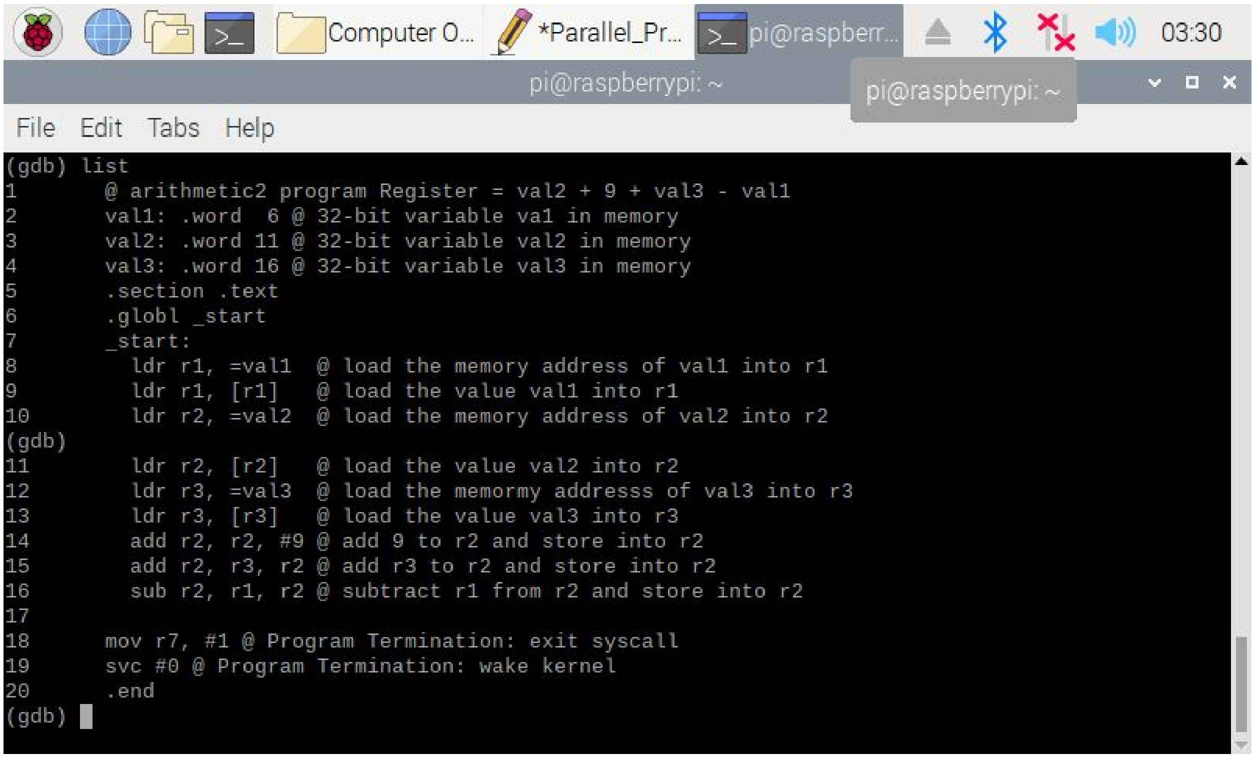
After setting the breakpoint on line 15, we run the program and use the memory command to examine the contents.



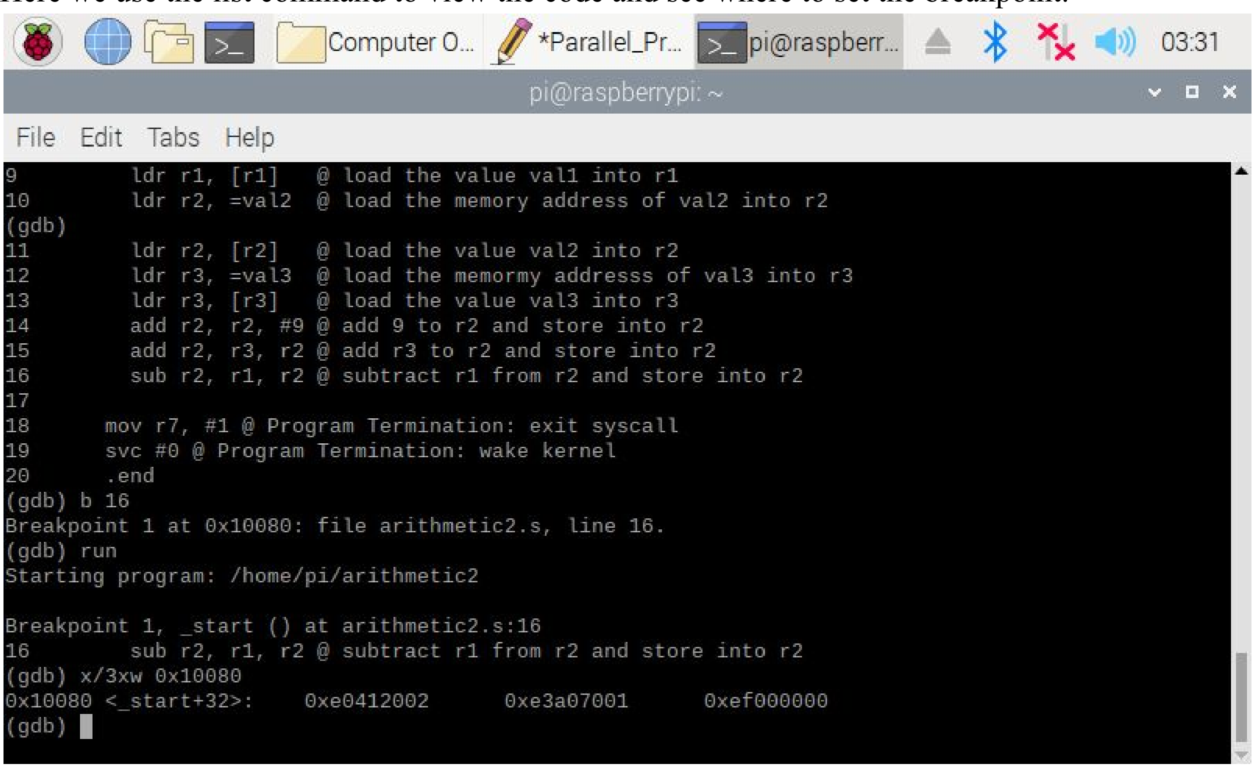
Code for arithemtic2



The commands used to create a file name for the program, and assemble link and run program. No output is shown in the terminal window, so we put the program in debug by using the GDB commands to examine the contents of the registers and memory. I had errors in my code because I forgot to place the '#' in front of the integer and didn't place it in the right spot.



Here we use the list command to view the code and see where to set the breakpoint.



After setting the breakpoint on line 16, we run the program and use the memory command to examine the contents.

Appendix

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

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

Youtube: https://youtu.be/hIdJnpbeZVQ