a) (46p) Foundation (reading material for this section is available at college, Week4, Project:

Assigenmnt2 folder, Introduction\_to\_Parallel\_Computing\_and\_the\_Raspberry\_Pi\_A2.pdf)

- (5p) Identifying the components on the raspberry PI B+

Display port; CPU/RAM; Ethernet Controller; USB port; Ethernet port; Power port; HDMI port.

- (5p) How many cores does the Raspberry Pi’s B+ CPU have

4.

- (8p) List three main differences between X86 (CISC) and ARM Raspberry PI (RISC). Justify you

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

1. instruction set:

x86 uses Complex Instruction Set, it has more operations and accesses complex instruction; ARM uses Reduced instruction set, it has simpler operations which allow the instructions to be executed more quickly, but it’s harder to write code at the same time.

1. Conditional Execution:

Almost all ARM instructions can be executed conditionally

1. Endian:

The endianness of x86 is little-endian, while the ARM is switchable endianness after version3.

- (6p) What is the difference between sequential and parallel computation and identify the practical

significance of each?

sequential: Instructions are executed sequentially one after another (on a single processor).

1.processor is heavily loaded, and the processors produce heat quickly. 2. Data in sequential processing transfers bit by bit.

parallel: Instructions from each part execute simultaneously on different processors.

1. take less time than sequential processing. 2. Cost more. 3. Data transfers in 8-bit form.

- (5p) 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, so the amount of available parallelism is proportional to the input size, leading to tremendous amounts of potential parallelism.

Task parallelism: The broad classification of task parallelism applies to solutions where parallelism is

organized around the functions to be performed rather than around the data.

- (6p) Explain the differences between processes and threads.

A process is the abstraction of a running program. Thread is a lightweight process that allow s a single executable /process to be decomposed to smaller, independent parts.

Processes do not share memory with each other, while the threads share the common memory of their process.

- (3p) What is OpenMP and what is OpenMP pragmas?

OpenMP is a low-level thread package which the library handles thread creation and management.

OpenMP pragmas are compiler directives that enable the compiler to generate threaded code.

- (4p) What applications benefit from multi-core (list four)?

Database servers; web servers; compilers; multimedia applications; scientific applications (CAD/CAM).

- (4p) Why Multicore? (why not single core, list four)

1. It is difficult to make single-core clock frequencies higher

2. There are deeply pipelined circuits problems: heat; speed of light; difficult design and verification; large design teams necessary; server farms need expensive AC.

3. Most of new applications require multithread.

4. general trend in computer architecture (shift towards more parallelism).

b)Parallel Programming Basics

original code:

1. #include <stdio.h>

2. #include <omp.h>

3. #include <stdlib.h

4. int main(int argc, char\*\* argv) {

5. int id, numThreads;

6. printf("\n");

7. if (argc > 1) {

8. omp\_set\_num\_threads( atoi(argv[1]) );

9. }

10. #pragma omp parallel

11. {

12. id = omp\_get\_thread\_num();

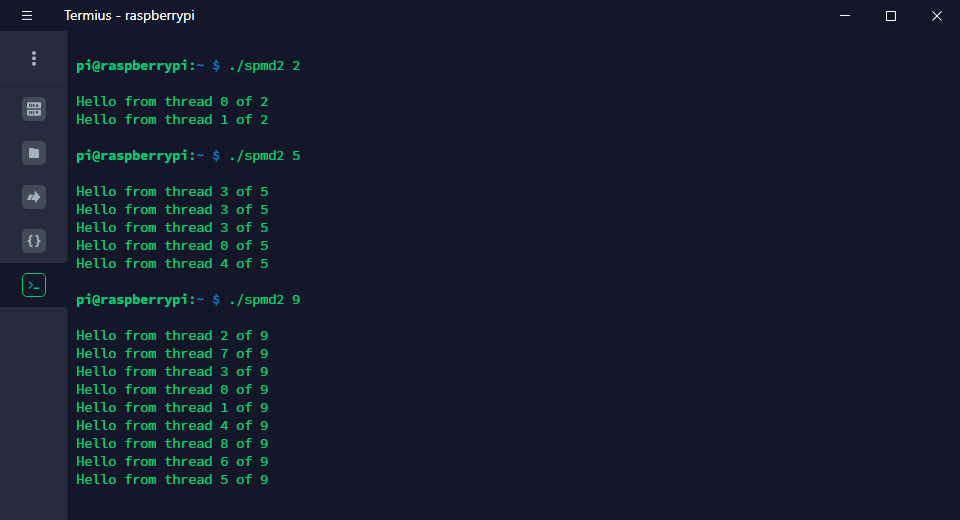
13. numThreads = omp\_get\_num\_threads();

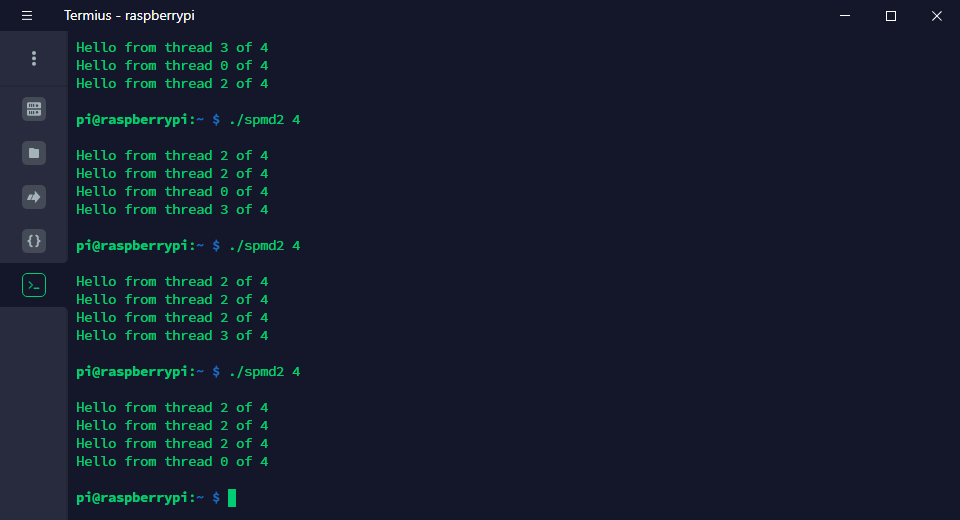
14. printf("Hello from thread %d of %d\n", id, numThreads);

15. }

16. printf("\n");

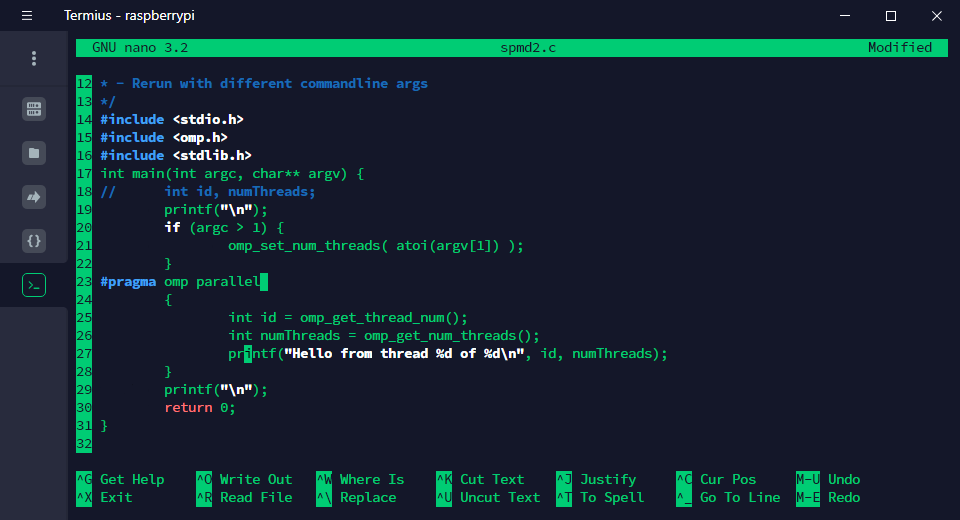
17. return 0;

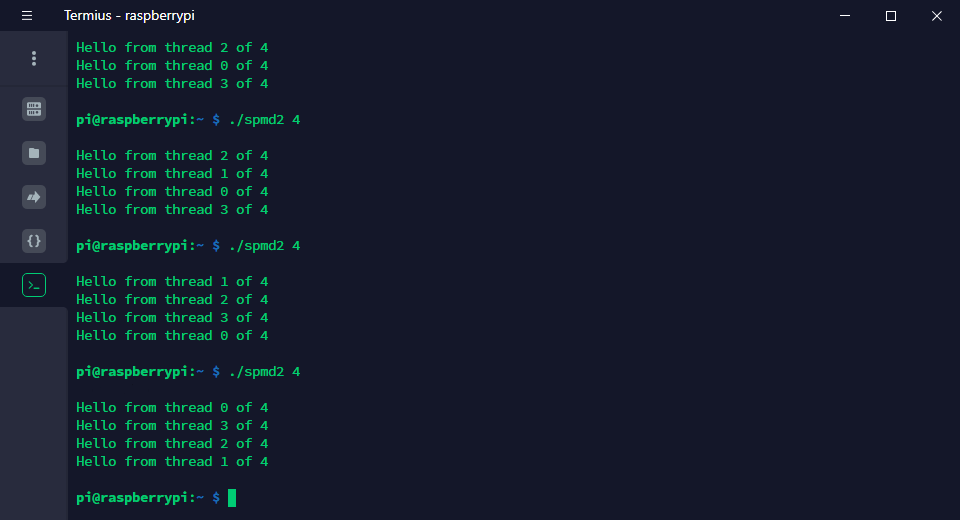
18.} 



I created the file spmd2.c according to the study guide and typed in the source code. After compiling with the gcc, I ran this program multiple times with 4 threads and then changed the command line parameters. I found that the thread id appears repeatedly in the one output.

I think this may be because the program does not declare the variables in the part after the fork.

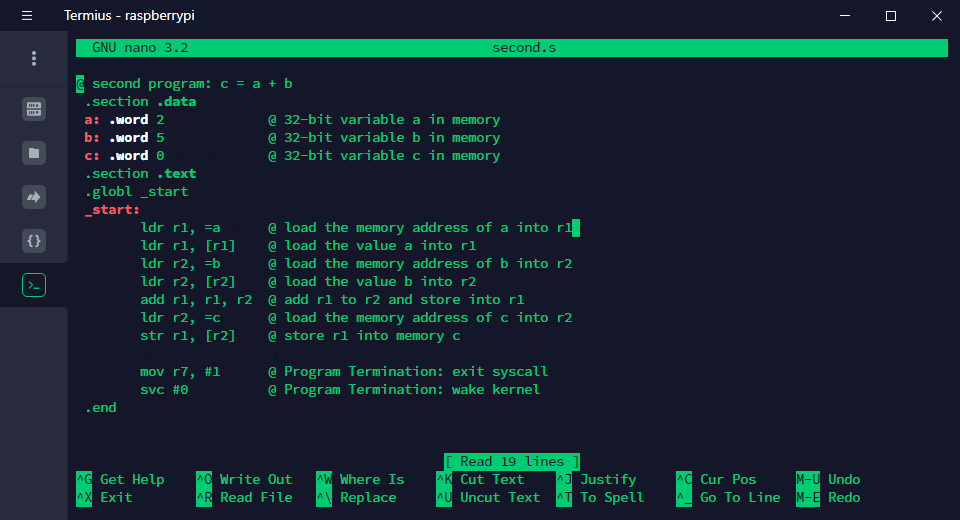


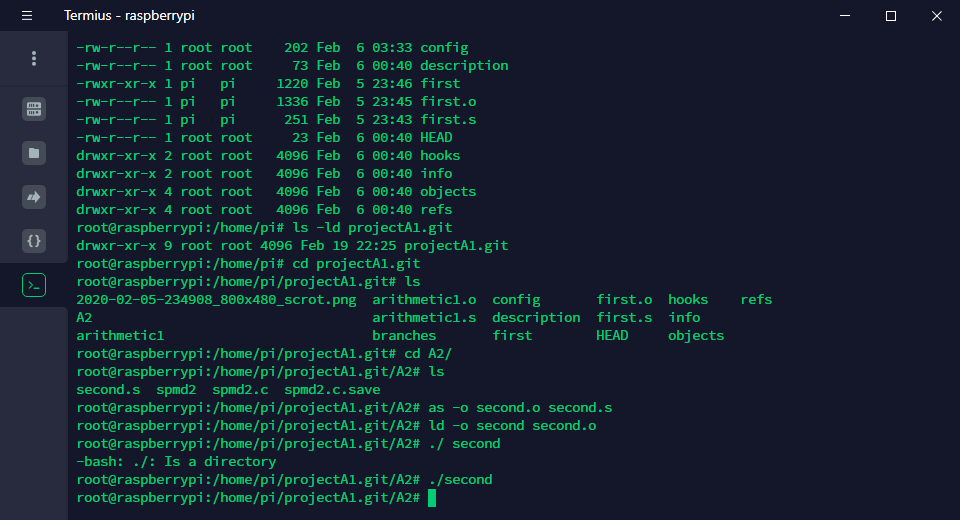


After changing the program according to the study guide, I ran the program again and found that each thread id was different in the output. This validates my previous hypothesis.

What I learned from this homework is how to use OpenMP to direct that the next block of code is run using multiple threads.

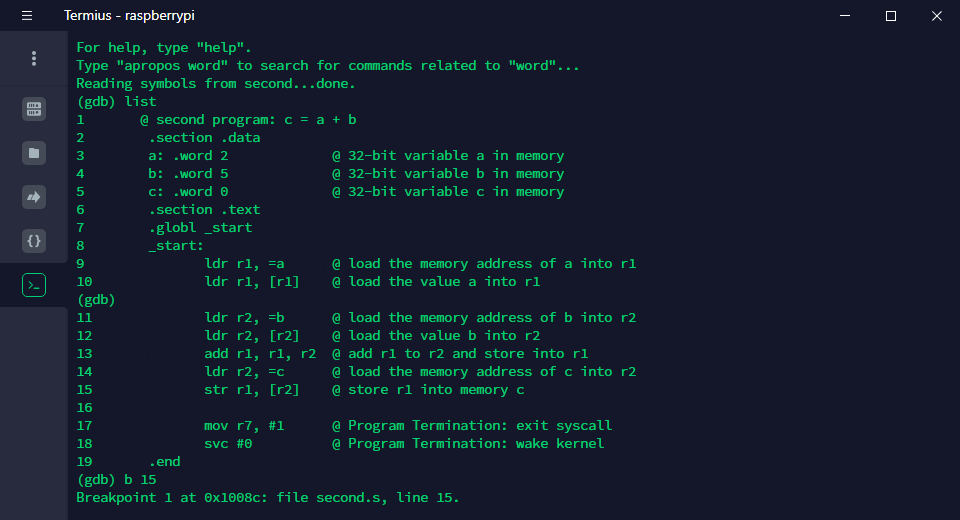
c)

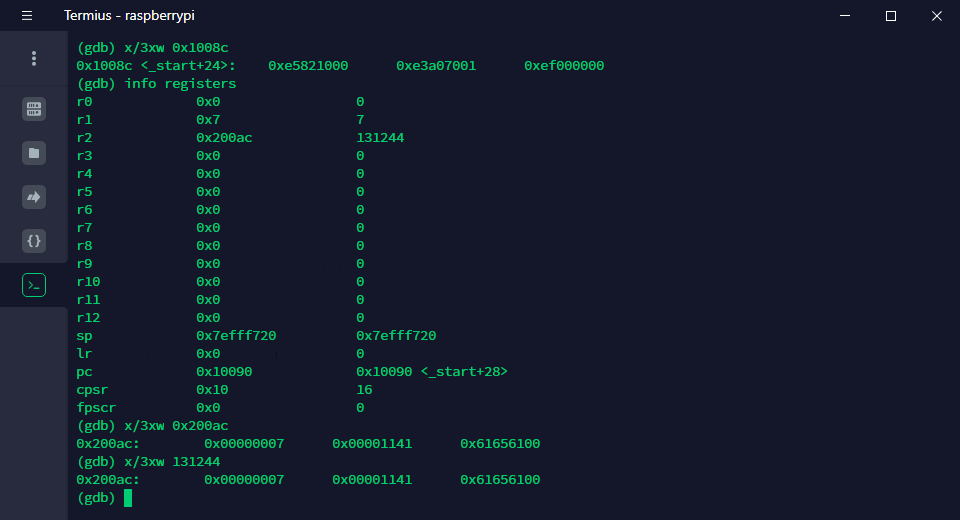
part1:

step6: 

there is no output, because there is no output statement in our program.

Step7:



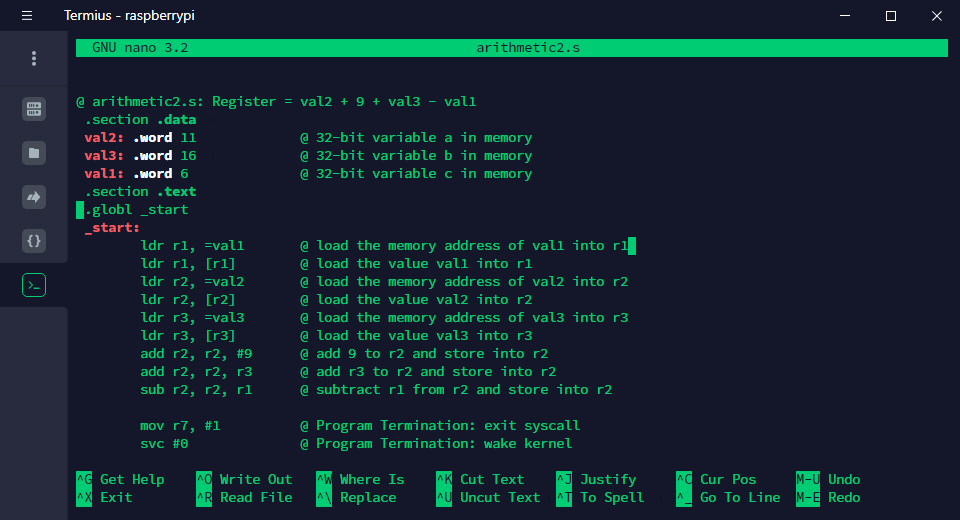


The output of the three words at the address of breakpoint 1 is like above.

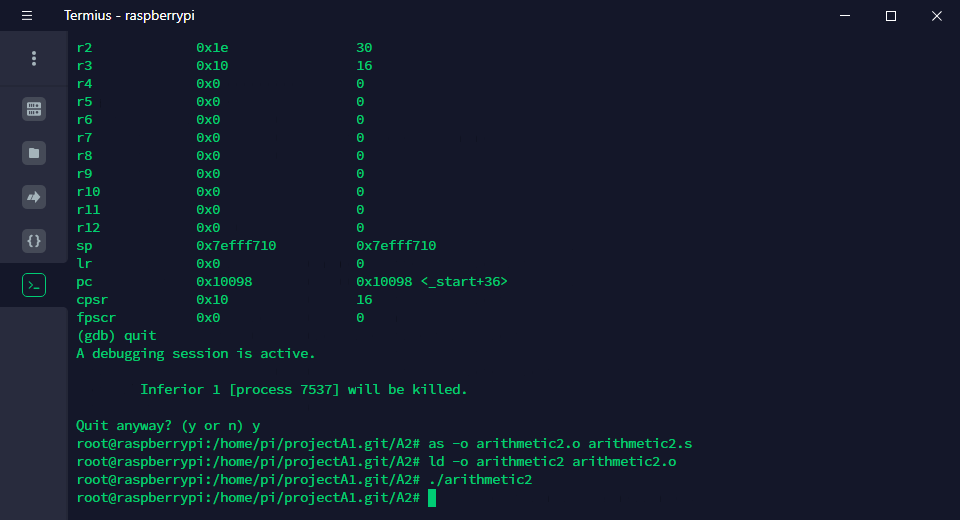
X(memory) / 3 (three) x (hex) w (word).

Part2:

Source code:

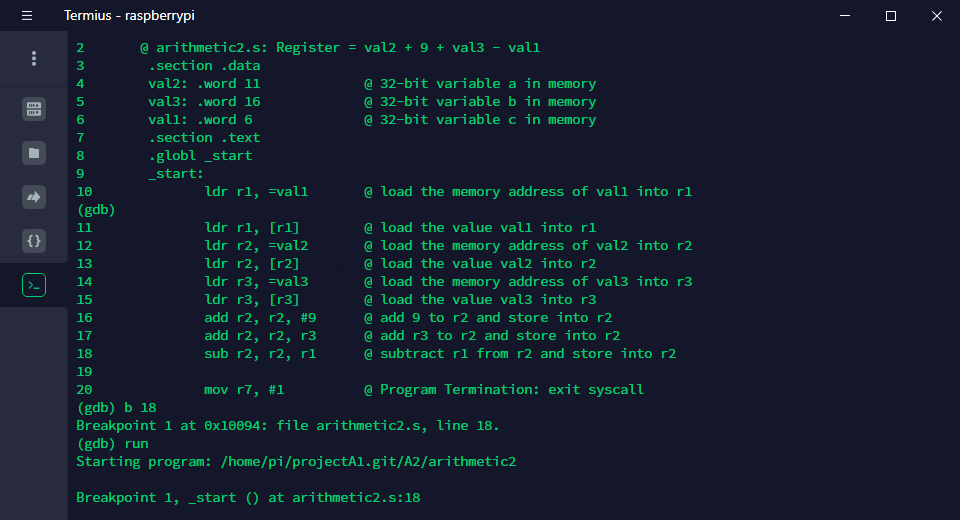


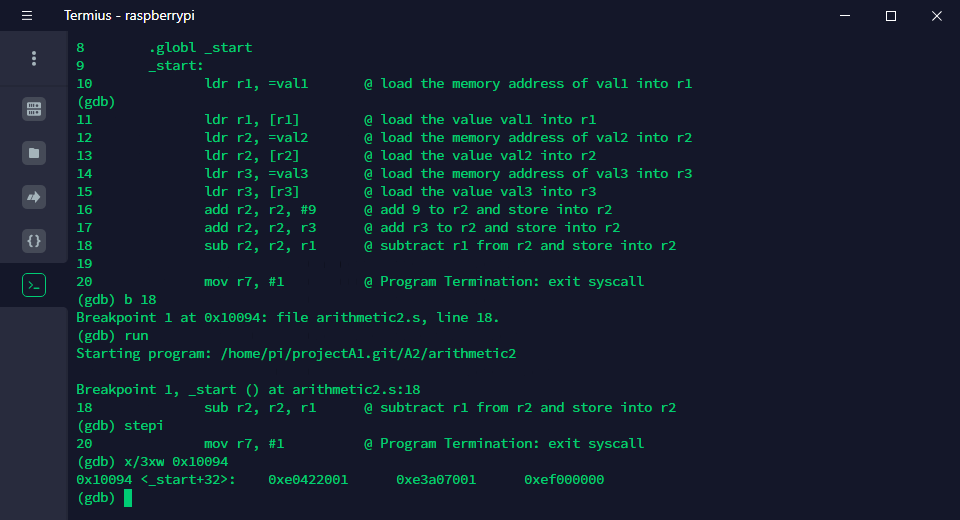
Simply Assemble, Link, run will not get any output because we don’t have any output command.



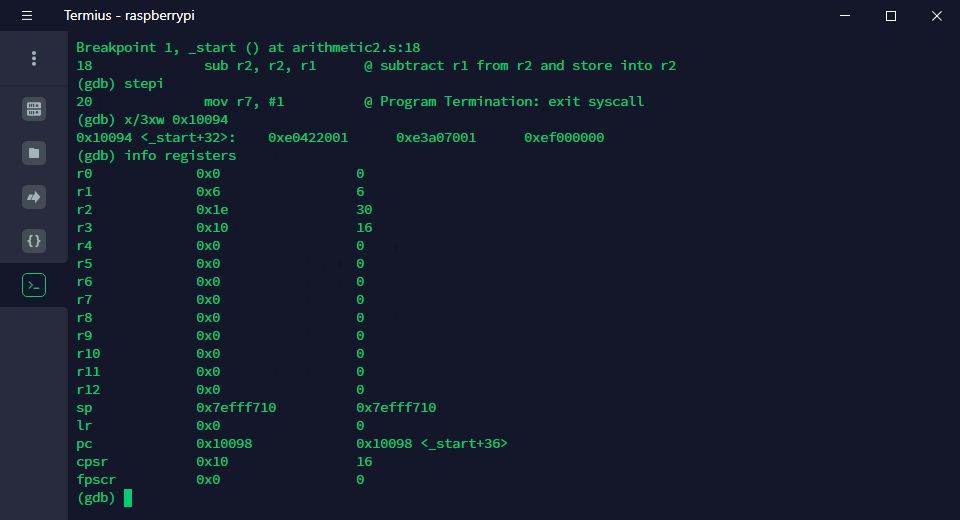
Now add **a flag “-g”** to the assembler command line and assemble, link, debug the program again.

I set the breakpoint at line 18:



The 3 words in hexadecimal starting at 0x10094: 

The value of the registers:



The value of r2 is the right answer (11+9+16-6=30), and we didn’t make any changes on r1 and r3. Thus, the result is correct.

Simply Assemble, Link, run will not get any output because we don’t have any output command.

