Developing Soft and Parallel Programming Skills Using Project-Based Learning

The Awad Winning Team

Spring 2019

Kishan Bhakta
Shilp Patel
Sean Staley
Mindy Tran
Donald Weaver

Planning and Scheduling –

Our team name is The Awad Winning Team, in honor of our fearless leader Dr. Awad Mussa. The planning and scheduling for this assignment first was fairly straightforward. We all chose our assigned tasks based on what we thought we would be best at, and there was no controversy when choosing. During the scheduling, we ran into a bit of a block, due to everyone leading such a busy schedule. We were able to agree on a common time, however, for everyone to meet and complete the group tasks. All of the planning and scheduling discussions took place in slack, which made communication between members very manageable. Below is a detailed table describing the different members tasks and obligations. The table was used to keep track of assignments during the project.

Name	Email	Task	Duration	Dependency	Due	Notes
Sean Staley	Sstaley4@student.gsu.edu	Report	3	Coordinating meeting time and communicating on slack	2/8/19	Make sure submissions are sent.
Mindy Tran	Mtran42@student.gsu.edu	Slack/ Keeping track of table	1	Keeping track of tasks	2/1/19	Make sure everyone writes introduction
Shilp Patel	Spatel225@student.gsu.edu	GitHub	3	Maintaining GitHub	2/4/19	Late for video
Kishan Bhakta	Kbhakta1@student.gsu.edu	Setting up raspberry pi/ Arm assembly programming	6	Pi Report	2/6/19	Take Screenshots
Donald Weaver	Dweaver20@student.gsu.edu	Task 6	4	Recording setup/ YouTube Maintenance.	2/8/19	Keep video length 3-8 mins.

Teamwork Basics –

Q. Work Norms: How will work be distributed? Who will set deadlines? What happens if someone doesn't follow through on his/her commitment (for example, misses a deadline)? How will the work be reviewed? What happens if people have different opinions about the quality of the work? What happens if people have different work habits (e.g., some people like to get assignments done right away; others work better with the pressure of a deadline).

A. The work was distributed by each member choosing what they think their strong-suit would be. If someone misses a deadline, they will receive either partial or no credit and the coordinator will finish their task. All of the team members opinions matter and we will work together to make sure all of our work is satisfactory to everyone. Work habits have to be adjusted for the group work. The important thing is getting the assignment done on time. We will all review the work on GitHub and make sure it is acceptable.

Q. Facilitator Norms: Will you use a facilitator? How will the facilitator be chosen? Will you rotate the position? What are the responsibilities of the facilitator? (see below)

A. The facilitator/ Coordinator is responsible for making sure everyone has a job and does their part on time.

Q. Communication Norms: When should communication takes place and through what medium (e.g., do some people prefer to communicate through e-mail while others would rather talk on the phone)?

A. The team has been communicating through slack, but we have all exchanged phone numbers as well.

Q. Meeting Norms: What is everyone's schedule? Should one person be responsible for coordinating meetings? Do people have a preference for when meetings are held? Where is a good place to hold meetings? What happens if people are late to a meeting? What happens if a group member misses a meeting? What if he/ she misses several meetings?

A. The meeting times have been the most difficult part of this assignment. All of us have full school loads and other obligations outside of school, not to mention commutes. We have coordinated the meetings as a team and have been very understanding towards everyone's responsibilities and obligations. If someone is late or unable to attend a meeting, there will be no hard feelings. If a group member attends no meetings or is continually late, we will be forced to give them partial credit for assignments that require us all to be together. The 5th floor study rooms in the library is where we have chosen to meet, which works for everyone.

Q. Consideration Norms: Can people eat at meetings? smoke? What happens if someone is dominating the discussion? How can norms be changed if someone is not comfortable with what is going on in the team?

A. Food is not an issue, and none of us smoke. We have not had a problem with anyone dominating the discussions, but if that were to happen anyone who feels uncomfortable or unheard can chime in and change the way the discussion is going. We are all adults, and able to have respectable dialogue without getting our feelings hurt.

Q. What to do to get the task accomplished and the team members' satisfaction high?

A. All members get their work done on time and be accountable for their part.

Q. Answer all the questions in the Work Norms, Facilitator Norms, Communication Norms using your own words and your own context. (Answered above)

Q. As a team, select two cases out of the four mentioned in Handling Difficult Behavior. (use your own words and your own context)

A. Case 1: Confront the person and ask them to change whatever is controversial.
Case 2: If case 1 has been performed unsuccessfully and the problem is ongoing, the team will sit the problem child down again, and inform them that we will not continue to work with him/her unless this behavior changes.

Q. When making decisions, If the team is having trouble reaching consensus, what should you do? (use your own words and your own context)

A. If the team is having trouble reaching a consensus, we will consider all other options that will make everyone happy. If none of those work, and at least 3 of 5 members agree, then we will vote and the other two will have to deal with disagreeing.

Q. What should you do if person may reach a decision more quickly than others and pressure people to move on before it is a good idea to do so?

A. We work and make decisions as a team, and all take input from everyone before a consensus is reached, which prevents this scenario from happening.

Q. What happens if most people on the team want to get an "A" on the assignment, but another person decides that a "B" will be acceptable

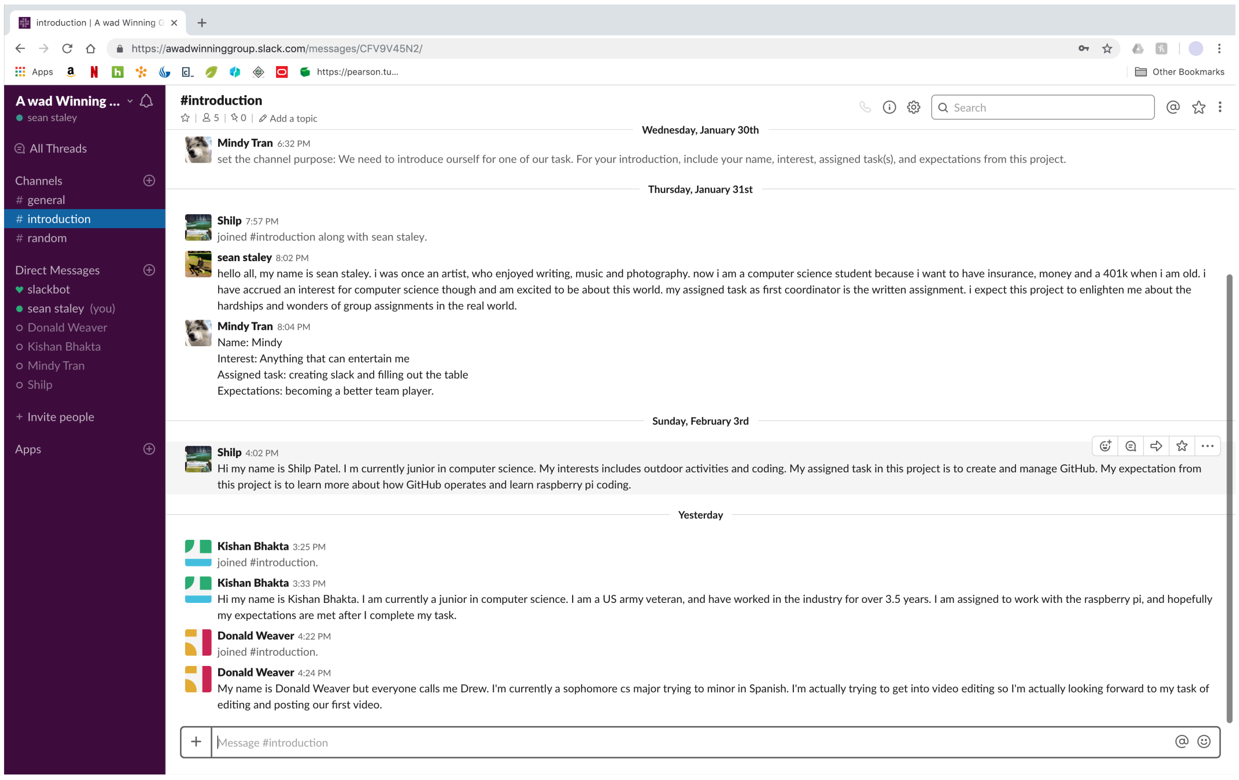
A. All of us want to do our best, and have not come upon this problem, but if we did, we would try to convince the person to do their best. If this person continually does not submit assignments, we will be forced to give them partial or no credit.

Raspberry PI Installation and ARM Assembly –

Kishan was responsible for handling the PI installation and ARM assembly for this project. He most certainly was the MVP of this assignment. His work with the PI and detailed lab report was outstanding. This report takes you step by step through his work with the PI as he creates and assembles files, uses assembly language to do arithmetic operations, links file to get an executable, and debugs the program. The PDF file written by him is attached along with this report.

Appendix –

Slack – awadwinninggroup.slack.com



Wednesday, January 30th

Mindy Tran 6:32 PM
set the channel purpose: We need to introduce ourself for one of our task. For your introduction, include your name, interest, assigned task(s), and expectations from this project.

Thursday, January 31st

Shilp 7:57 PM
joined #introduction along with sean staley.

sean staley 8:02 PM
hello all, my name is sean staley. i was once an artist, who enjoyed writing, music and photography. now i am a computer science student because i want to have insurance, money and a 401k when i am old. i have accrued an interest for computer science though and am excited to be about this world. my assigned task as first coordinator is the written assignment. i expect this project to enlighten me about the hardships and wonders of group assignments in the real world.

Mindy Tran 8:04 PM
Name: Mindy
Interest: Anything that can entertain me
Assigned task: creating slack and filling out the table
Expectations: becoming a better team player.

Sunday, February 3rd

Shilp 4:02 PM
Hi my name is Shilp Patel. I m currently junior in computer science. My interests includes outdoor activities and coding. My assigned task in this project is to create and manage GitHub. My expectation from this project is to learn more about how GitHub operates and learn raspberry pi coding.

Yesterday

Kishan Bhakta 3:25 PM
joined #introduction.

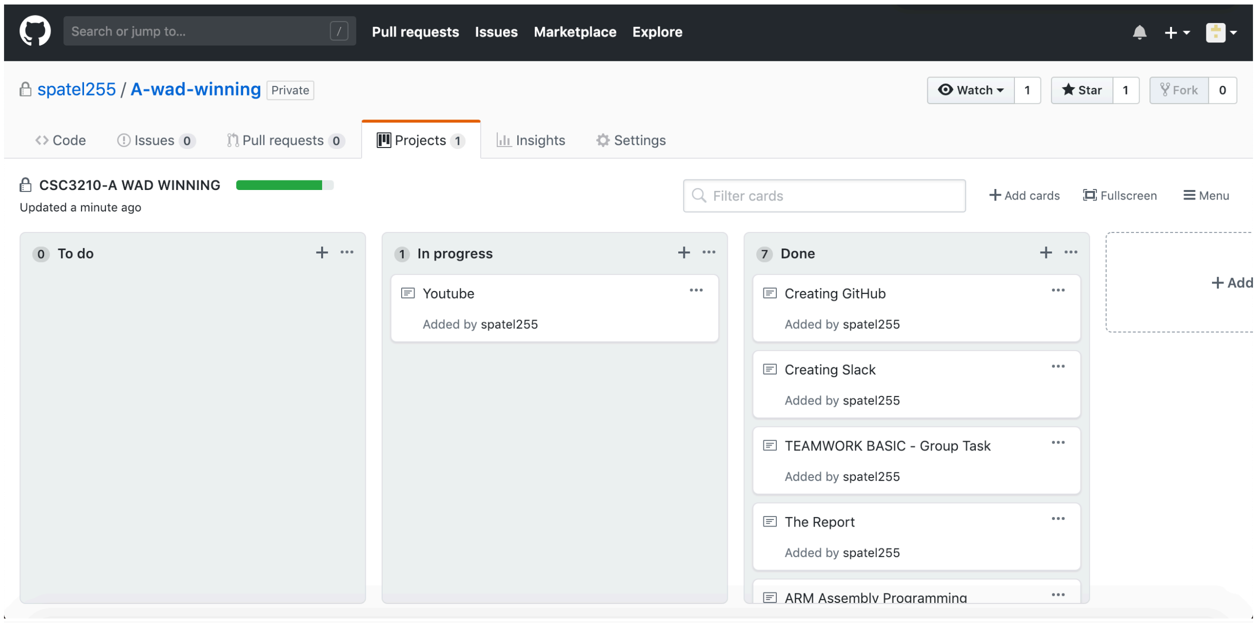
Kishan Bhakta 3:33 PM
Hi my name is Kishan Bhakta. I am currently a junior in computer science. I am a US army veteran, and have worked in the industry for over 3.5 years. I am assigned to work with the raspberry pi, and hopefully my expectations are met after I complete my task.

Donald Weaver 4:22 PM
joined #introduction.

Donald Weaver 4:24 PM
My name is Donald Weaver but everyone calls me Drew. I'm currently a sophomore cs major trying to minor in Spanish. I'm actually trying to get into video editing so I'm actually looking forward to my task of editing and posting our first video.

Message #introduction

GitHub – https://github.com/spatel255/A-wad-winning/projects/1

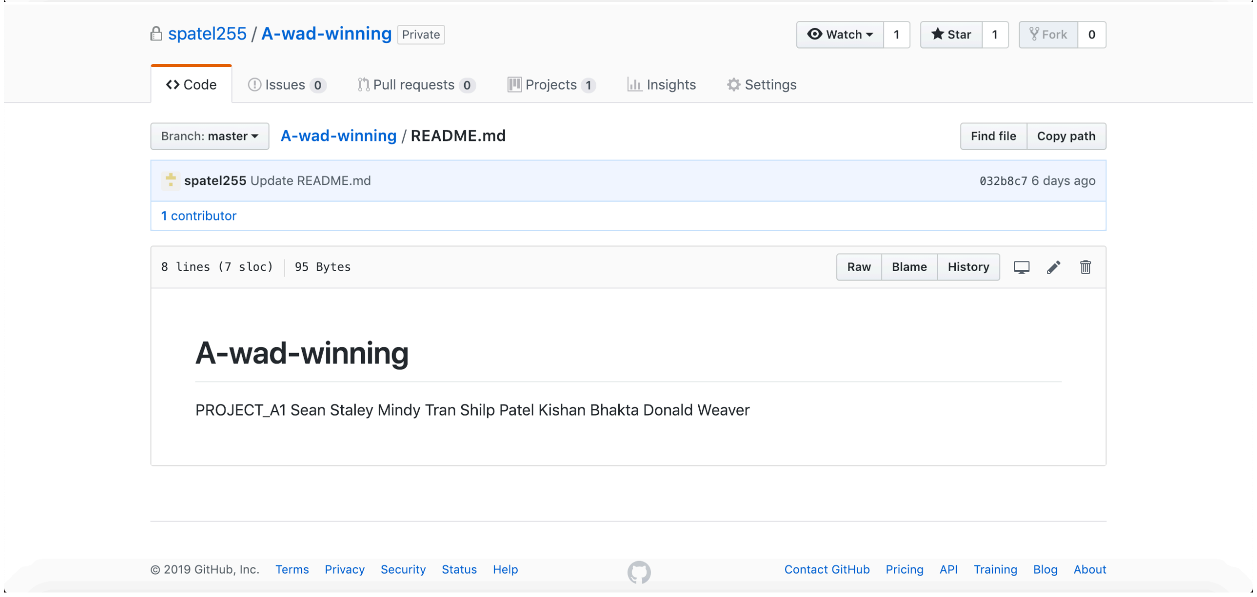


CSC3210-A WAD WINNING
Updated a minute ago

0 To do

1 In progress
- Youtube — Added by spatel255

7 Done
- Creating GitHub — Added by spatel255
- Creating Slack — Added by spatel255
- TEAMWORK BASIC - Group Task — Added by spatel255
- The Report — Added by spatel255
- ARM Assembly Programming



A-wad-winning / README.md

spatel255 Update README.md 032b8c7 6 days ago

1 contributor

8 lines (7 sloc) | 95 Bytes

A-wad-winning

PROJECT_A1 Sean Staley Mindy Tran Shilp Patel Kishan Bhakta Donald Weaver

© 2019 GitHub, Inc. Terms Privacy Security Status Help Contact GitHub Pricing API Training Blog About

Raspberry Pi Installation and ARM Assembly Programming

Objective:

- Downloading and installing Raspbian Operating System image on MicroSD
- Setup Raspberry Pi to connect with Monitor and I/O device (i.e. keyboard, mouse, WIFI-router/internet)
- Learn to:
  - Create file
  - Assemble file
  - Link file to get an executable
  - Debug
  - Write assembly code to perform arithmetic expressions

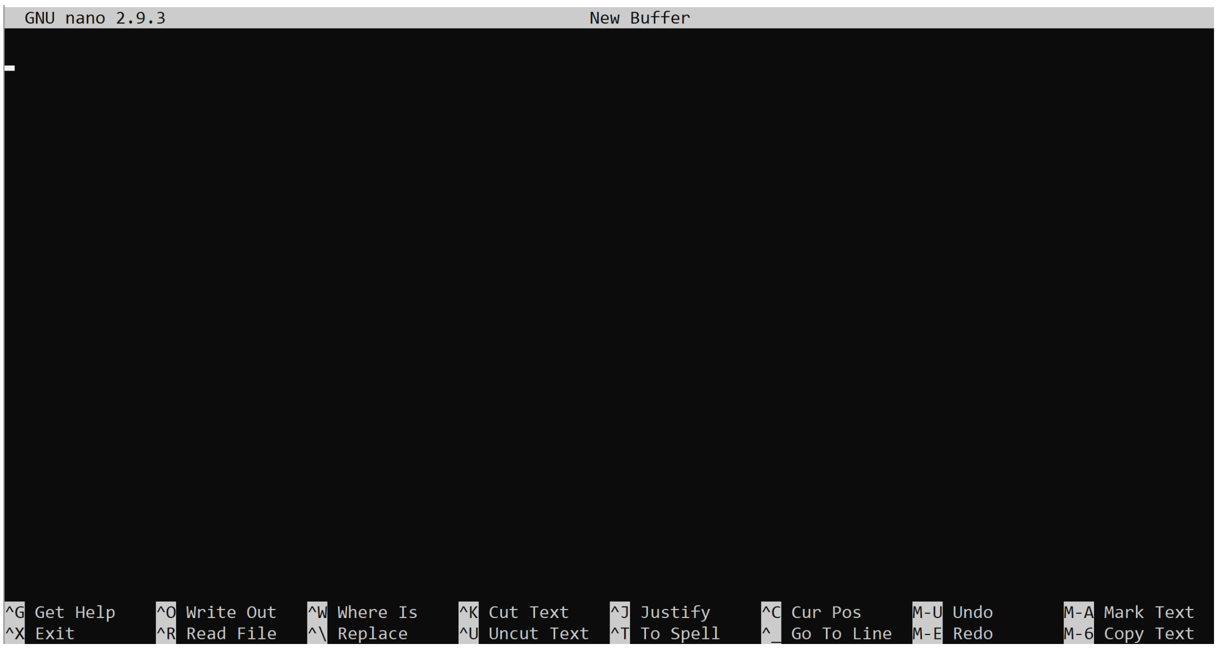
Part 1: First Program

Steps:

1. Open terminal and type the following command in the terminal window and press enter.

   `nano first.s`

   - This command allows the user to create a file called `first.s`, which is an assembly source code file type, and edit the file in Nano, the built-in text editor in the terminal.
   - You will see:

```
  GNU nano 2.9.3                                    New Buffer

^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos    M-U Undo      M-A Mark Text
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text ^T To Spell   ^_ Go To Line M-E Redo      M-6 Copy Text
```

2. Write following program in nano and save (Ctrl w)

```
@first program
 .section .data
 .section .text
 .globl _start
 _start:
    mov r1,#5     @load r1 with 5
    sub r1,r1,#1  @subtract 1 from r1
    add r1,r1,#4  @add 4 to r1

    mov r7,#4     @Program Termination: exit syscall
    svc #0        @Program Termination: wake kernel
 .end
```

3. The **assembler** converts assembly code into machine code and produces object files as output, just like a compiler does.
    - Type the following code in the terminal to assemble `first.s` file:

      `as -o first.o first.s`
    - This command tells the assembler to create an objective file called first.o from first.s source code file created in step 2.
4. The **linker** provides some platform-specific code to be put inside it, and access to libraries. When a program has multiple files, the linker combines these files into an executable program. It is also responsible for arranging the objects in a programs address space.
    - Type the following code in the terminal to link `first.o` file and get an executable.

      `ld -o first first.o`
    - If everything goes well you will get an executable file (.exe) in the directory you are currently in.
5. Now its time to run the program, by typing the following command into the terminal:

   `./first`
    - *Observation*: There was not an output.
    - Reason: For most of the programs in the manipulations of data between CPU registers and memory, will not have an output because there is no use of I/O devices.
    - Solution: GNU Debugger(**GDB**) can step through the program and examine the contents of the registers and memory.
        - First you preserve the symbols and line numbers of the source code by adding a `-g` flag to the assembler command from step 3.

          `as -g -o first.o first.s`

      Note: The linker command stays the same as step 4.
        - To launch GDB type gdb followed by the executable file name into the terminal.

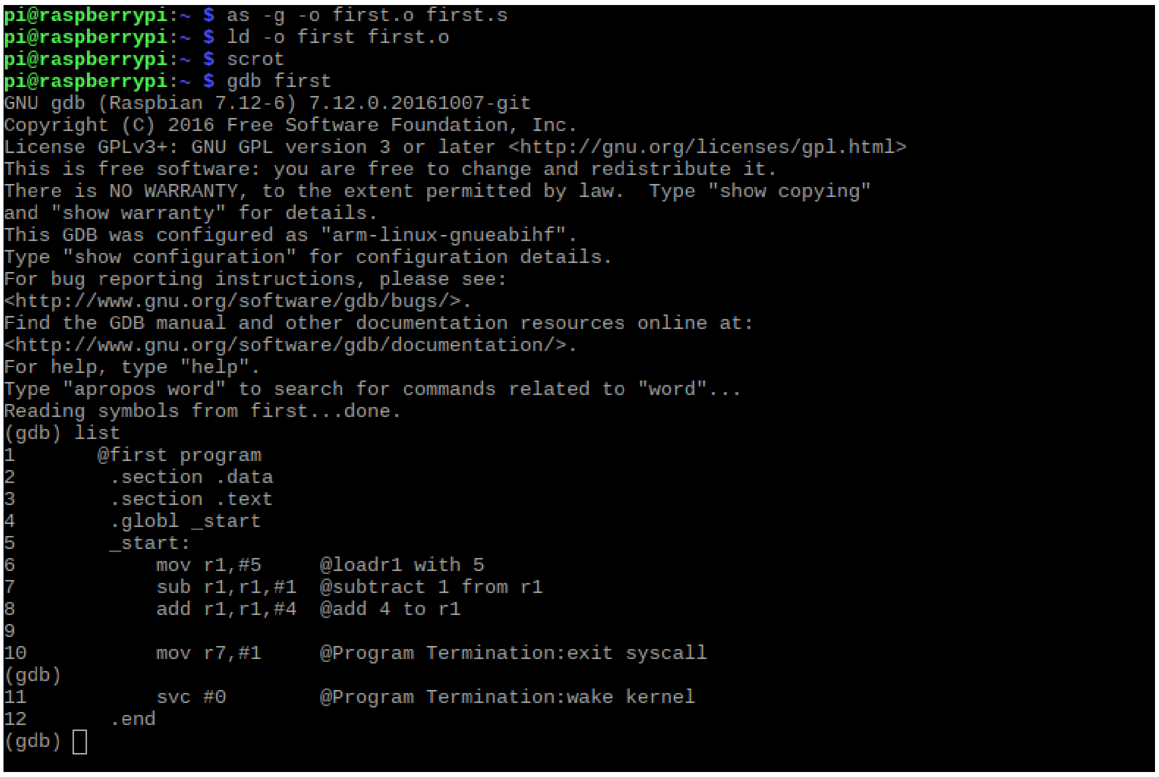
          `gdb first`

          Note: look for `(gdb)` in the terminal prompt.
        - To list the source code, type the following command in the debugger prompt:

          `list`

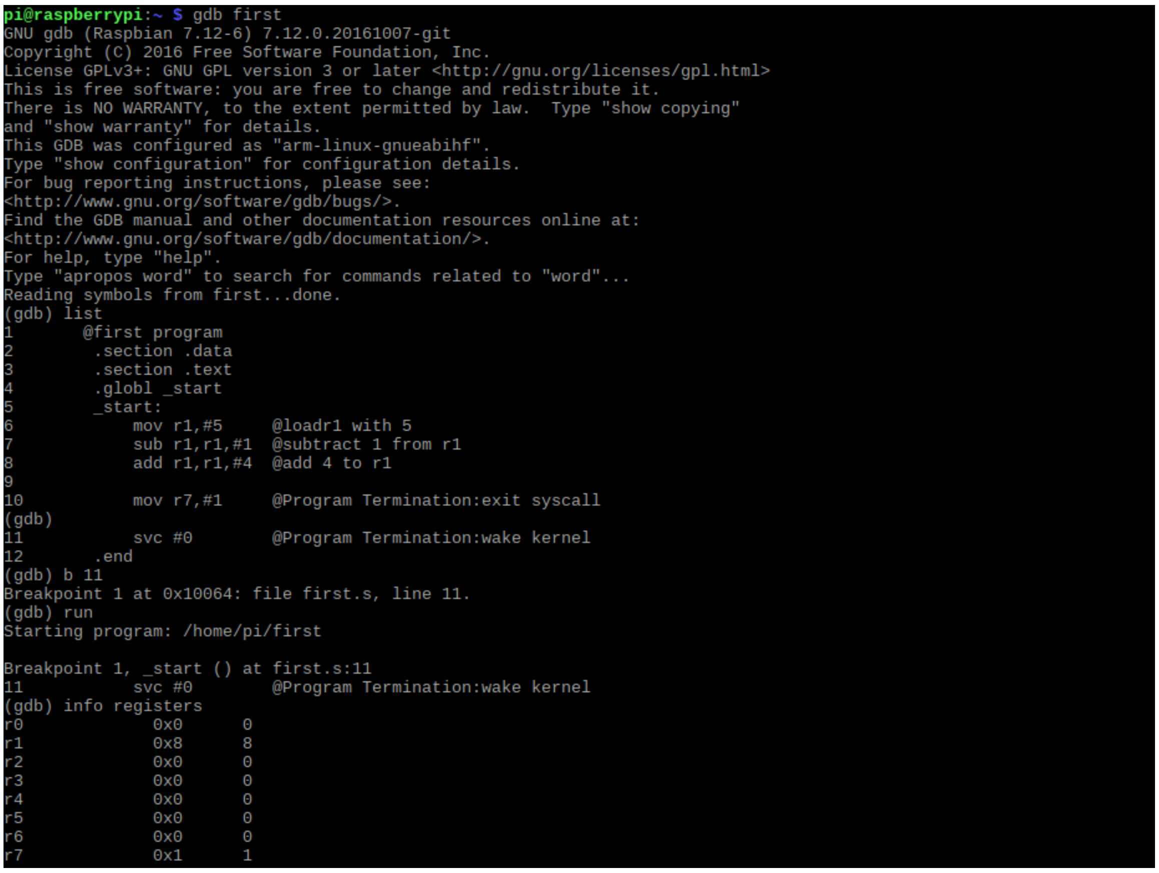
Note: The list command list the first ten lines of the source code, to see the rest just press Enter.

- You will see:



```
pi@raspberrypi:~ $ as -g -o first.o first.s
pi@raspberrypi:~ $ ld -o first first.o
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ gdb first
GNU gdb (Raspbian 7.12-6) 7.12.0.20161007-git
Copyright (C) 2016 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 first...done.
(gdb) list
1       @first program
2        .section .data
3        .section .text
4        .globl _start
5        _start:
6            mov r1,#5     @loadr1 with 5
7            sub r1,r1,#1  @subtract 1 from r1
8            add r1,r1,#4  @add 4 to r1
9
10           mov r7,#1     @Program Termination:exit syscall
(gdb)
11           svc #0        @Program Termination:wake kernel
12       .end
(gdb) 
```

- To gain access to each the states of each register, you have to stop the program is its execution ***right before*** the set breakpoints. To implement a breakpoint you enter break (alias: *b*) and the line number in which you would like to set the breakpoint.



```
pi@raspberrypi:~ $ gdb first
GNU gdb (Raspbian 7.12-6) 7.12.0.20161007-git
Copyright (C) 2016 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 first...done.
(gdb) list
1       @first program
2        .section .data
3        .section .text
4        .globl _start
5        _start:
6            mov r1,#5     @loadr1 with 5
7            sub r1,r1,#1  @subtract 1 from r1
8            add r1,r1,#4  @add 4 to r1
9
10           mov r7,#1     @Program Termination:exit syscall
(gdb)
11           svc #0        @Program Termination:wake kernel
12       .end
(gdb) b 11
Breakpoint 1 at 0x10064: file first.s, line 11.
(gdb) run
Starting program: /home/pi/first

Breakpoint 1, _start () at first.s:11
11           svc #0        @Program Termination:wake kernel
(gdb) info registers
r0             0x0      0
r1             0x8      8
r2             0x0      0
r3             0x0      0
r4             0x0      0
r5             0x0      0
r6             0x0      0
r7             0x1      1
```

> Abstract:
>
> - Ran `gdb first`, and entered the debugger prompt.
> - Ran `(gdb) list` to list the source code, 10 lines at a time.
> - Ran `(gdb) b 11` to set breakpoint at line 11.
> - Ran `(gdb) run` to run the program.
> - Ran `(gdb) info registers` to verify if the program stored the results in the right registers. The program was executed in the debugger and stopped execution before line 11, where the break was instructed to occur. The value of 1 from the instructions listed in line 10, was stored in r7. The value of 8 in the arithmetic function listed in line 8 was stored in r1.

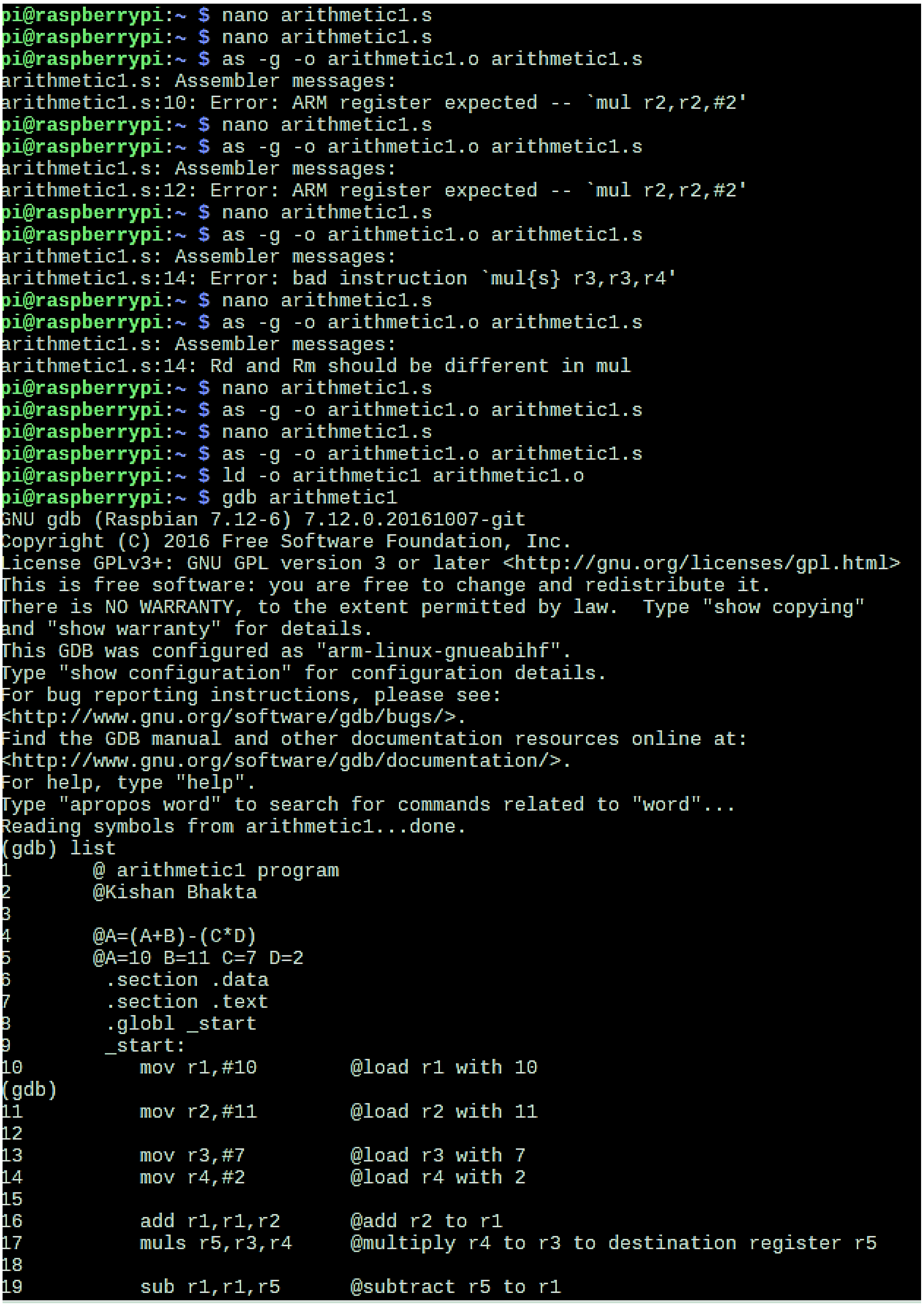
Part 2: Arithmetic1 Program

Objective:

- Write a program that calculated the following expression, `A = (A + B) - (C * D)` without declaring or using memory variable and only using registers.
- Assemble, link, run and debug the program.

Process:

- The assembly, link, and running the program steps are identical to Part 1. These are a few errors that occurred when writing the source code to the executable file:



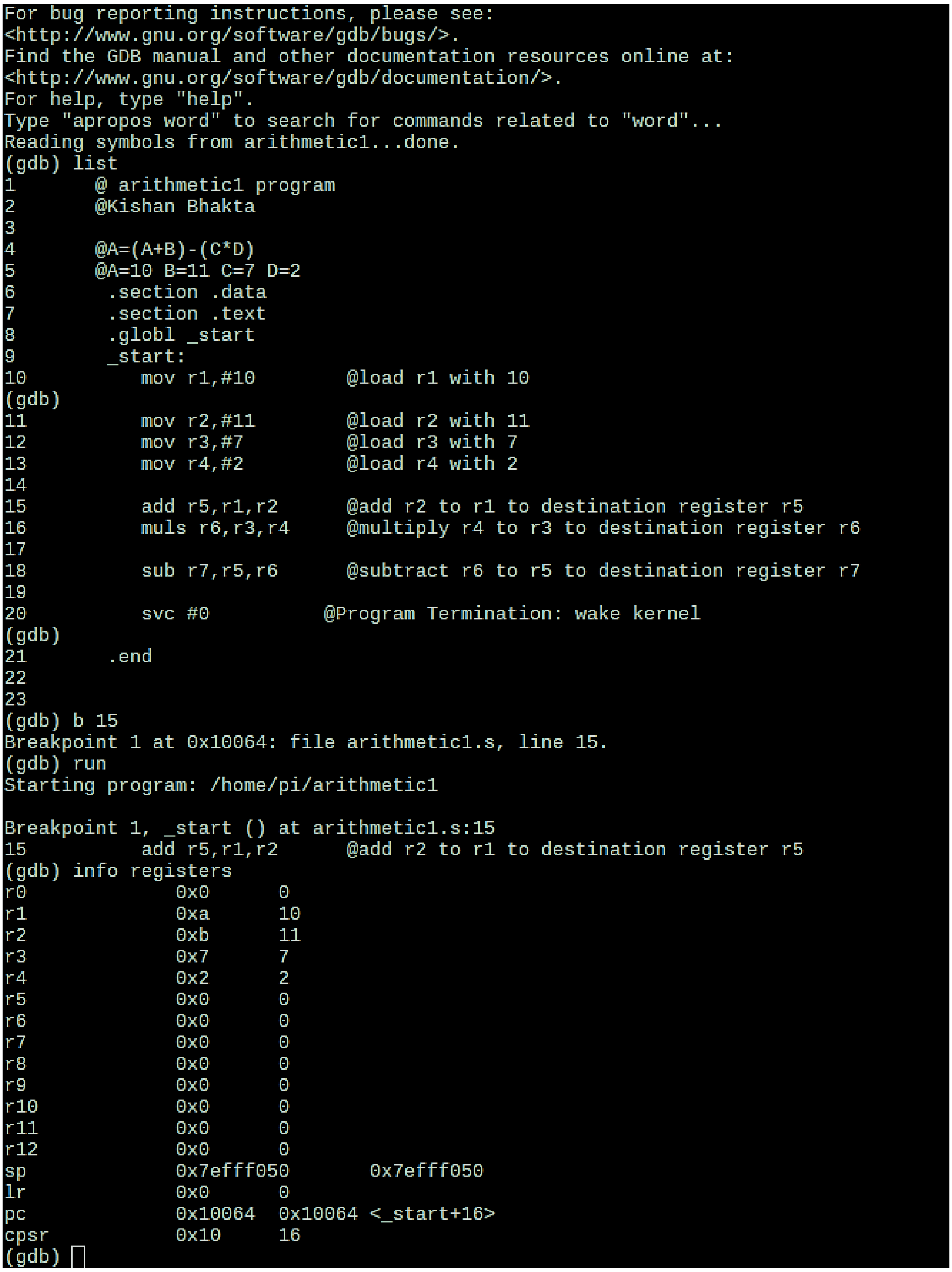
```
pi@raspberrypi:~ $ nano arithmetic1.s
pi@raspberrypi:~ $ nano arithmetic1.s
pi@raspberrypi:~ $ as -g -o arithmetic1.o arithmetic1.s
arithmetic1.s: Assembler messages:
arithmetic1.s:10: Error: ARM register expected -- `mul r2,r2,#2'
pi@raspberrypi:~ $ nano arithmetic1.s
pi@raspberrypi:~ $ as -g -o arithmetic1.o arithmetic1.s
arithmetic1.s: Assembler messages:
arithmetic1.s:12: Error: ARM register expected -- `mul r2,r2,#2'
pi@raspberrypi:~ $ nano arithmetic1.s
pi@raspberrypi:~ $ as -g -o arithmetic1.o arithmetic1.s
arithmetic1.s: Assembler messages:
arithmetic1.s:14: Error: bad instruction `mul{s} r3,r3,r4'
pi@raspberrypi:~ $ nano arithmetic1.s
pi@raspberrypi:~ $ as -g -o arithmetic1.o arithmetic1.s
arithmetic1.s: Assembler messages:
arithmetic1.s:14: Rd and Rm should be different in mul
pi@raspberrypi:~ $ nano arithmetic1.s
pi@raspberrypi:~ $ as -g -o arithmetic1.o arithmetic1.s
pi@raspberrypi:~ $ nano arithmetic1.s
pi@raspberrypi:~ $ as -g -o arithmetic1.o arithmetic1.s
pi@raspberrypi:~ $ ld -o arithmetic1 arithmetic1.o
pi@raspberrypi:~ $ gdb arithmetic1
GNU gdb (Raspbian 7.12-6) 7.12.0.20161007-git
Copyright (C) 2016 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 arithmetic1...done.
(gdb) list
1       @ arithmetic1 program
2       @Kishan Bhakta
3
4       @A=(A+B)-(C*D)
5       @A=10 B=11 C=7 D=2
6        .section .data
7        .section .text
8        .globl _start
9        _start:
10          mov r1,#10        @load r1 with 10
(gdb)
11          mov r2,#11        @load r2 with 11
12
13          mov r3,#7         @load r3 with 7
14          mov r4,#2         @load r4 with 2
15
16          add r1,r1,r2      @add r2 to r1
17          muls r5,r3,r4     @multiply r4 to r3 to destination register r5
18
19          sub r1,r1,r5      @subtract r5 to r1
```

> Observations:
>
> - Errors were being throw at line 17, where the multiplication instructions needed to execute.

> Solution:
>
> - Multiplication instruction cannot execute if another register is being multiplied to the register, therefore I had multiply r4 to r3 and load r5 with the result.

- After I fixed my errors and was able to execute the command `gdb arithmetic1` in the terminal. I was able to list my source code in the GNU Debugger, and set breakpoints to see the state of each register at each breakpoint. First breakpoint was set at line 15 by executing the following command in the GNU debugger prompt:

  `(gdb) b 15`



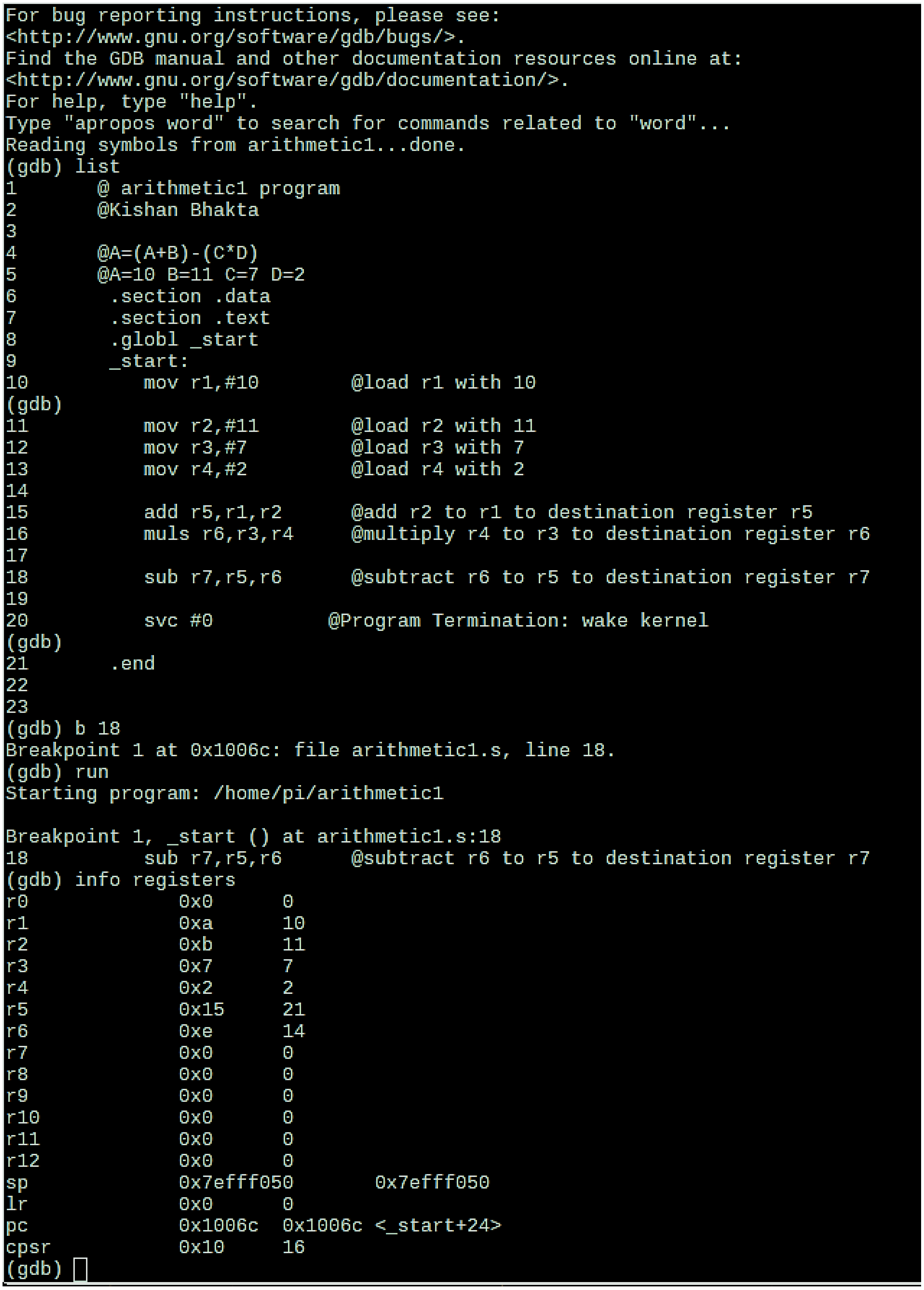
```
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from arithmetic1...done.
(gdb) list
1       @ arithmetic1 program
2       @Kishan Bhakta
3
4       @A=(A+B)-(C*D)
5       @A=10 B=11 C=7 D=2
6        .section .data
7        .section .text
8        .globl _start
9        _start:
10          mov r1,#10        @load r1 with 10
(gdb)
11          mov r2,#11        @load r2 with 11
12          mov r3,#7         @load r3 with 7
13          mov r4,#2         @load r4 with 2
14
15          add r5,r1,r2      @add r2 to r1 to destination register r5
16          muls r6,r3,r4     @multiply r4 to r3 to destination register r6
17
18          sub r7,r5,r6      @subtract r6 to r5 to destination register r7
19
20          svc #0          @Program Termination: wake kernel
(gdb)
21       .end
22
23
(gdb) b 15
Breakpoint 1 at 0x10064: file arithmetic1.s, line 15.
(gdb) run
Starting program: /home/pi/arithmetic1

Breakpoint 1, _start () at arithmetic1.s:15
15          add r5,r1,r2      @add r2 to r1 to destination register r5
(gdb) info registers
r0             0x0      0
r1             0xa      10
r2             0xb      11
r3             0x7      7
r4             0x2      2
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             0x7efff050       0x7efff050
lr             0x0      0
pc             0x10064  0x10064 <_start+16>
cpsr           0x10     16
(gdb)
```

- This image shows the state of the loaded registers with the values assigned to them:
  - r1 = 10
  - r2 = 11
  - r3 = 7
  - r4 = 2

- The second breakpoint was set at line 18 by executing the following command in the GNU debugger prompt:

  `(gdb) b 18`

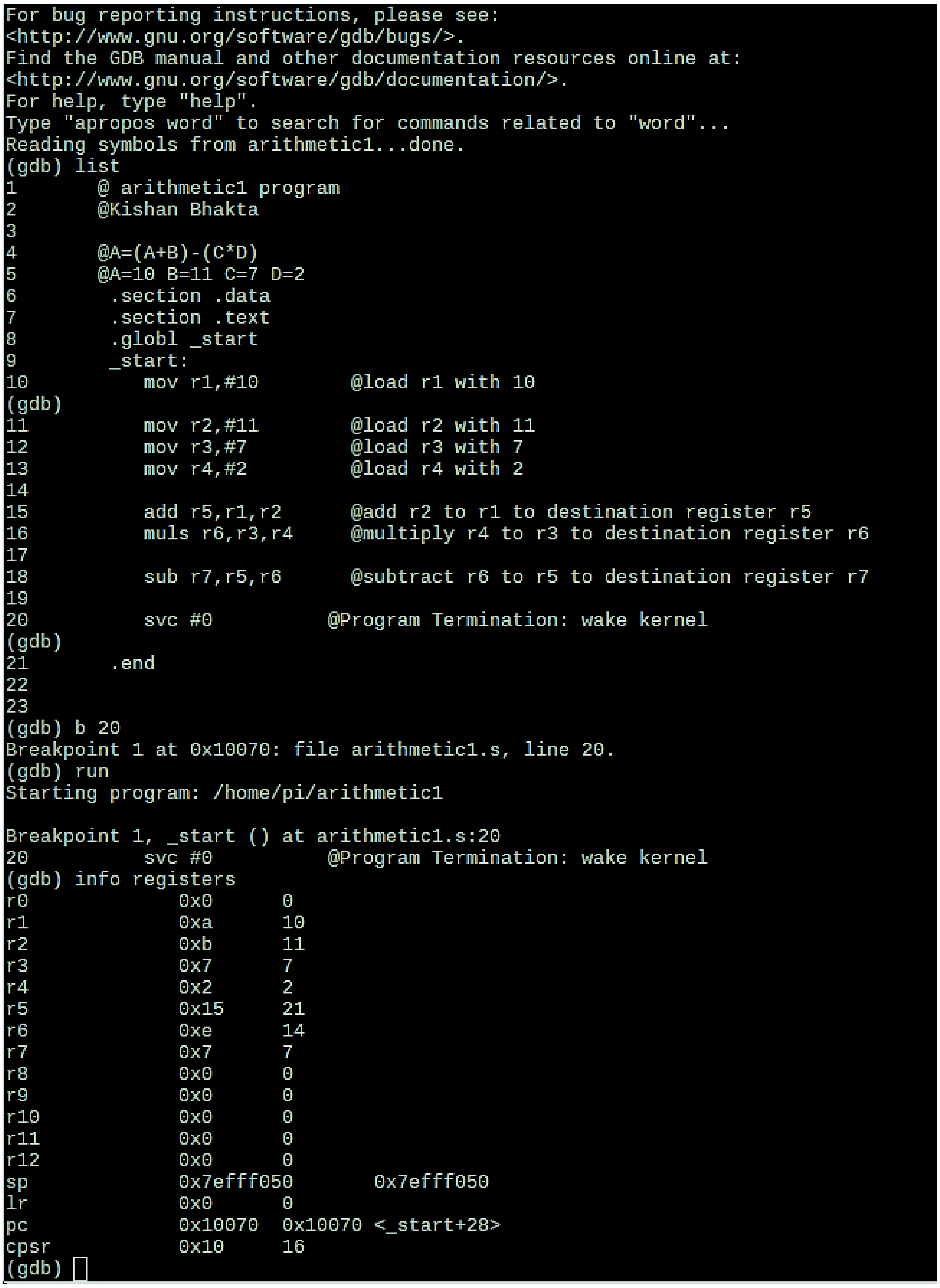


```
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from arithmetic1...done.
(gdb) list
1       @ arithmetic1 program
2       @Kishan Bhakta
3
4       @A=(A+B)-(C*D)
5       @A=10 B=11 C=7 D=2
6        .section .data
7        .section .text
8        .globl _start
9        _start:
10          mov r1,#10        @load r1 with 10
(gdb)
11          mov r2,#11        @load r2 with 11
12          mov r3,#7         @load r3 with 7
13          mov r4,#2         @load r4 with 2
14
15          add r5,r1,r2      @add r2 to r1 to destination register r5
16          muls r6,r3,r4     @multiply r4 to r3 to destination register r6
17
18          sub r7,r5,r6      @subtract r6 to r5 to destination register r7
19
20          svc #0          @Program Termination: wake kernel
(gdb)
21       .end
22
23
(gdb) b 18
Breakpoint 1 at 0x1006c: file arithmetic1.s, line 18.
(gdb) run
Starting program: /home/pi/arithmetic1

Breakpoint 1, _start () at arithmetic1.s:18
18          sub r7,r5,r6      @subtract r6 to r5 to destination register r7
(gdb) info registers
r0             0x0      0
r1             0xa      10
r2             0xb      11
r3             0x7      7
r4             0x2      2
r5             0x15     21
r6             0xe      14
r7             0x0      0
r8             0x0      0
r9             0x0      0
r10            0x0      0
r11            0x0      0
r12            0x0      0
sp             0x7efff050       0x7efff050
lr             0x0      0
pc             0x1006c  0x1006c <_start+24>
cpsr           0x10     16
(gdb) 
```

    - This image shows the state of the loaded registers with the values of the add and mul ARM instruction:
        - r1 = 10
        - r2 = 11
        - r3 = 7
        - r4 = 2
        - r5 = 21 [***add r5, r1,r2***]
        - r6 = 14 [***mul r6, r3,r4***]
- The third breakpoint was set at line 20 by executing the following command in the GNU debugger prompt:

  `(gdb) b 20`



```
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 arithmetic1...done.
(gdb) list
1       @ arithmetic1 program
2       @Kishan Bhakta
3
4       @A=(A+B)-(C*D)
5       @A=10 B=11 C=7 D=2
6        .section .data
7        .section .text
8        .globl _start
9        _start:
10          mov r1,#10        @load r1 with 10
(gdb)
11          mov r2,#11        @load r2 with 11
12          mov r3,#7         @load r3 with 7
13          mov r4,#2         @load r4 with 2
14
15          add r5,r1,r2      @add r2 to r1 to destination register r5
16          muls r6,r3,r4     @multiply r4 to r3 to destination register r6
17
18          sub r7,r5,r6      @subtract r6 to r5 to destination register r7
19
20          svc #0          @Program Termination: wake kernel
(gdb)
21       .end
22
23
(gdb) b 20
Breakpoint 1 at 0x10070: file arithmetic1.s, line 20.
(gdb) run
Starting program: /home/pi/arithmetic1

Breakpoint 1, _start () at arithmetic1.s:20
20          svc #0          @Program Termination: wake kernel
(gdb) info registers
r0             0x0      0
r1             0xa      10
r2             0xb      11
r3             0x7      7
r4             0x2      2
r5             0x15     21
r6             0xe      14
r7             0x7      7
r8             0x0      0
r9             0x0      0
r10            0x0      0
r11            0x0      0
r12            0x0      0
sp             0x7efff050       0x7efff050
lr             0x0      0
pc             0x10070  0x10070 <_start+28>
cpsr           0x10     16
(gdb)
```

- This image shows the state of the loaded registers with the values of the instruction listed above as well as the result loaded in r7.
    - r1 = 10
    - r2 = 11

- r3 = 7
- r4 = 2
- r5 = 21 [***add r5, r1,r2***]
- r6 = 14 [***mul r6, r3,r4***]
- r7 = 7 [***sub r7, r5, r6***]