

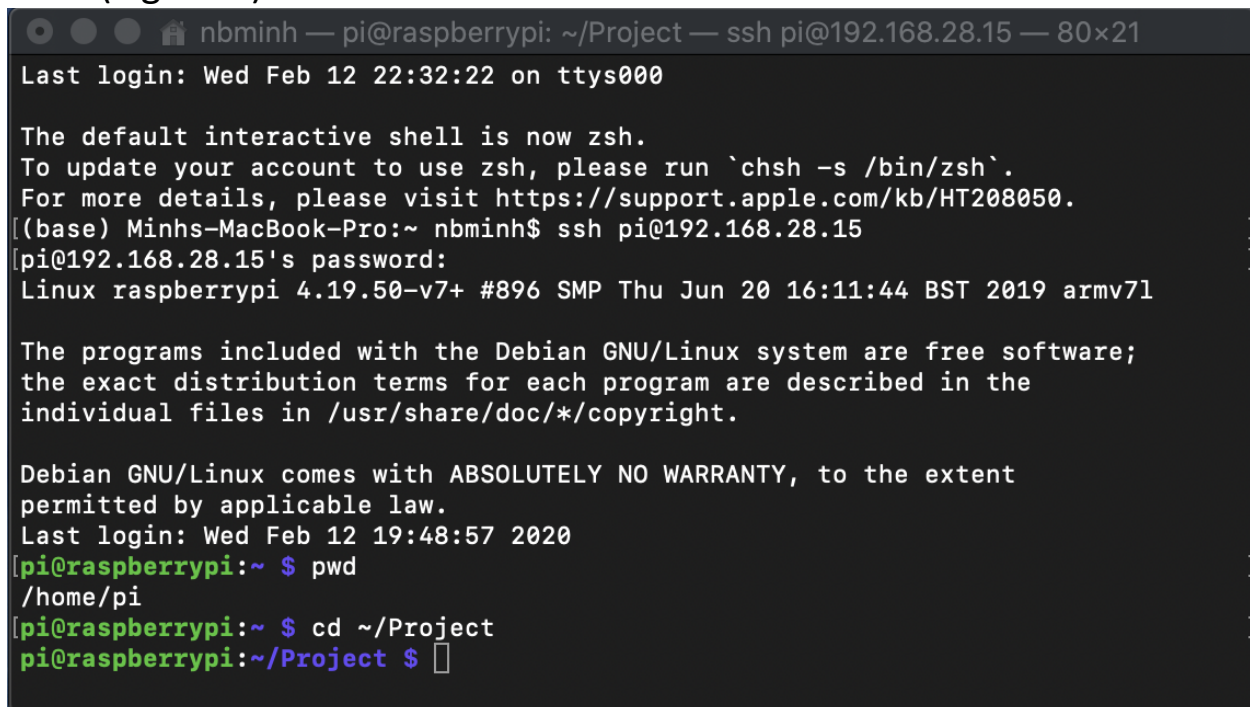
Name: Minh Binh Nguyen
PantherID: 002-46-4288

Project 2 – Task 4

ARM Assembly Programming

Part 1

I started with getting my Raspberry connected to my computer and redirected to “Project” folder where I store all my Project files for this class (Figure 1).



```
nbminh — pi@raspberrypi: ~/Project — ssh pi@192.168.28.15 — 80x21
Last login: Wed Feb 12 22:32:22 on ttys000

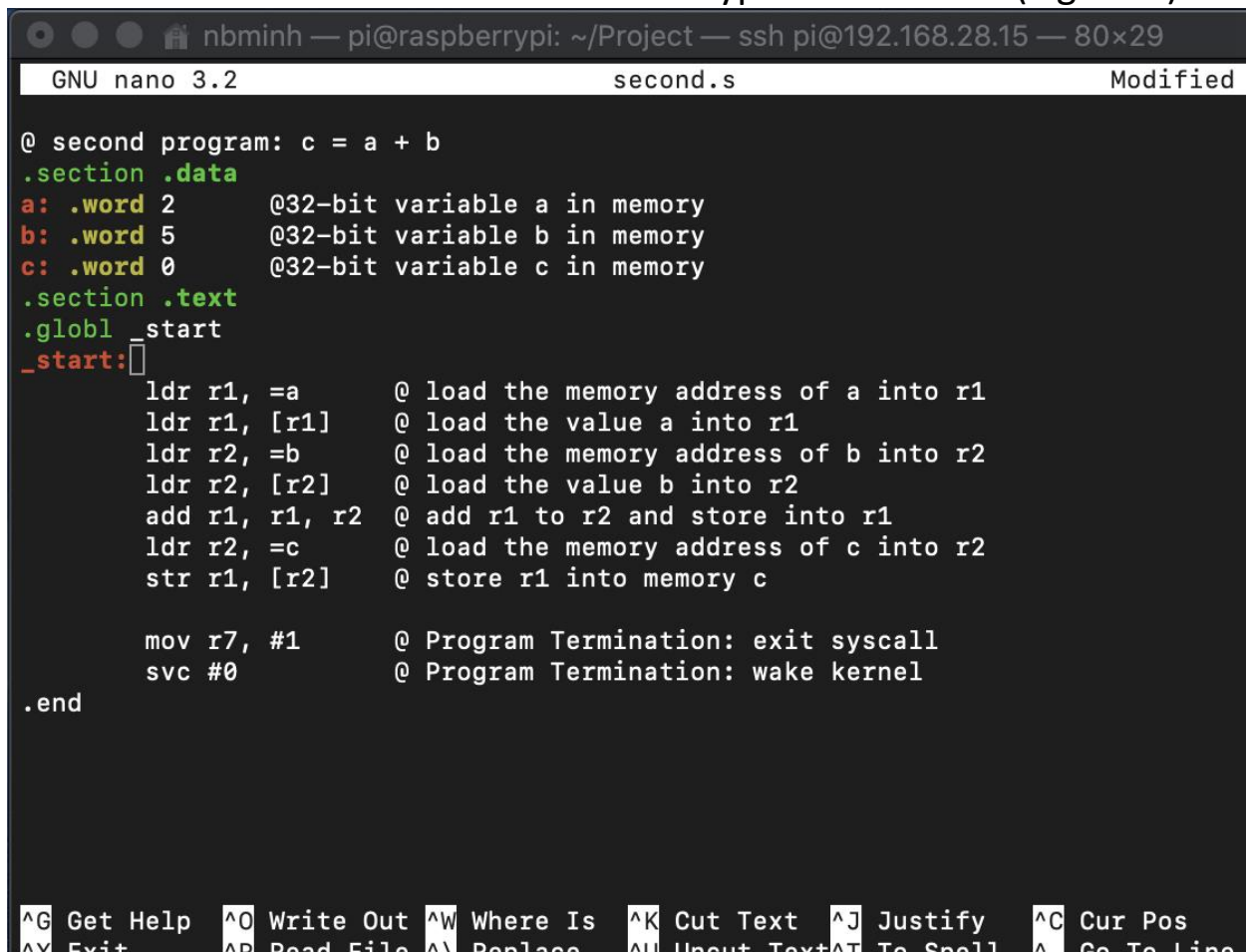
The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
[(base) Minhs-MacBook-Pro:~ nbminh$ ssh pi@192.168.28.15
pi@192.168.28.15's password:
Linux raspberrypi 4.19.50-v7+ #896 SMP Thu Jun 20 16:11:44 BST 2019 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Feb 12 19:48:57 2020
pi@raspberrypi:~ $ pwd
/home/pi
pi@raspberrypi:~ $ cd ~/Project
pi@raspberrypi:~/Project $
```

Figure 1

Then I created a file called `second.s` and typed in the code (Figure 2).



```
nbminh — pi@raspberrypi: ~/Project — ssh pi@192.168.28.15 — 80x29
GNU nano 3.2                                second.s                                Modified

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

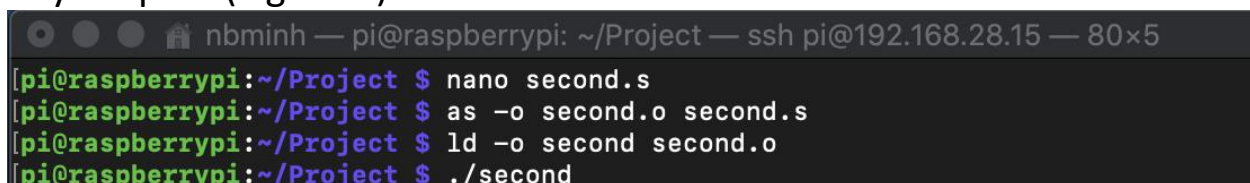
    mov r7, #1      @ Program Termination: exit syscall
    svc #0          @ Program Termination: wake kernel

.end

^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^Y Exit      ^P Read File ^\ Replace   ^U Uncut Text ^T To Spell  ^_ Go To Line
```

Figure 2

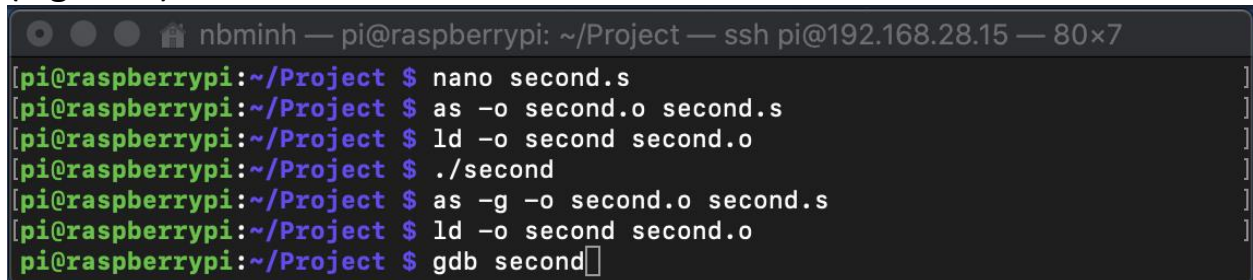
Then I assembled and linked the file to get an executable file (without using the flag “-g”). When running it, I didn’t see any output. This is to be expected because the code doesn’t have any instructions to produce any outputs (Figure 3).



```
nbminh — pi@raspberrypi: ~/Project — ssh pi@192.168.28.15 — 80x5
[pi@raspberrypi:~/Project $ nano second.s
[pi@raspberrypi:~/Project $ as -o second.o second.s
[pi@raspberrypi:~/Project $ ld -o second second.o
[pi@raspberrypi:~/Project $ ./second
```

Figure 3

Then I re-assembled and re-linked the file again (this time adding the “-g” flag for debug purpose), then use the GDB to debug the program (Figure 4).



```
nbminh — pi@raspberrypi: ~/Project — ssh pi@192.168.28.15 — 80x7
[pi@raspberrypi:~/Project $ nano second.s
[pi@raspberrypi:~/Project $ as -o second.o second.s
[pi@raspberrypi:~/Project $ ld -o second second.o
[pi@raspberrypi:~/Project $ ./second
[pi@raspberrypi:~/Project $ as -g -o second.o second.s
[pi@raspberrypi:~/Project $ ld -o second second.o
[pi@raspberrypi:~/Project $ gdb second
```

Figure 4

Here is the code being listed by the gdb. I set a breakpoint at line 9 and started debugging it line by line using stepi to understand what is going on with the registers and the memory (Figure 5).

```
nbminh — pi@raspberrypi: ~/Project — ssh pi@192.168.43.160 — 80x46
[pi@raspberrypi:~/Project $ 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-gnueabi".
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
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)
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 kernel
19     .end
20
[(gdb) b 9
Breakpoint 1 at 0x10078: file second.s, line 10.
[(gdb) run
Starting program: /home/pi/Project/second

Breakpoint 1, _start () at second.s:10
10         ldr r1, [r1]       @ load the value a into r1
```

Figure 5

After loading the address of a into r1, I can see the actual address being stored in r1 using the “info registers” command. It’s 0x200a4. By using the examining command “x/1xw 0x200a4”, I can see 1 item word in hex at that address. And it holds the value 2h which is 2d as expected (Figure 6).

```
nbminh — pi@raspberrypi: ~/Project — ssh pi@192.168.43.160 — 80x27
Breakpoint 1, _start () at second.s:10
10      ldr r1, [r1]    @ load the value a into r1
[(gdb) info registers
r0          0x0          0
r1          0x200a4      131236
r2          0x0          0
r3          0x0          0
r4          0x0          0
r5          0x0          0
r6          0x0          0
r7          0x0          0
r8          0x0          0
r9          0x0          0
r10         0x0          0
r11         0x0          0
r12         0x0          0
sp          0x7efff650    0x7efff650
lr          0x0          0
pc          0x10078      0x10078 <_start+4>
cpsr       0x10          16
fpscr      0x0          0
[(gdb) x/1dw
Argument required (starting display address).
[(gdb) x/1dw 0x200a4
0x200a4:      2
[(gdb) x/1xw 0x200a4
0x200a4:      0x00000002
```

Figure 6

After executing line 10, the value of a has been stored into r1 register (Figure 7)

```
nbminh — pi@raspberrypi: ~/Project — ssh pi@192.168.43.160 — 80x24
[(gdb) stepi
11          ldr r2, =b          @ load the memory address of b into r2
[(gdb) info registers
r0          0x0                0
r1          0x2                2
r2          0x0                0
r3          0x0                0
r4          0x0                0
r5          0x0                0
r6          0x0                0
r7          0x0                0
r8          0x0                0
r9          0x0                0
r10         0x0                0
r11         0x0                0
r12         0x0                0
sp          0x7efff650         0x7efff650
lr          0x0                0
pc          0x1007c            0x1007c <_start+8>
cpsr       0x10                16
fpscr      0x0                0
[(gdb) x/1xw 0x200a4
0x200a4:    0x00000002
[(gdb) stepi
```

Figure 7

Executing line 11 gives me the address of b in r2 register. The address is 0x200a8 which is a 4-byte step from address of a. This is because a is a 4-byte word. By examining this address, I got the value of b in the memory (which is 5h, also 5d) as expected (Figure 8).

```
nbminh — pi@raspberrypi: ~/Project — ssh pi@192.168.43.160 — 80x24
[(gdb) stepi
12          ldr r2, [r2]      @ load the value b into r2
[(gdb) info registers
r0          0x0              0
r1          0x2              2
r2          0x200a8          131240
r3          0x0              0
r4          0x0              0
r5          0x0              0
r6          0x0              0
r7          0x0              0
r8          0x0              0
r9          0x0              0
r10         0x0              0
r11         0x0              0
r12         0x0              0
sp          0x7efff650       0x7efff650
lr          0x0              0
pc          0x10080          0x10080 <_start+12>
cpsr       0x10             16
fpscr      0x0              0
[(gdb) x/1xw 0x200a8
0x200a8:    0x00000005
[(gdb) stepi
```

Figure 8

I keep stepping line by line and getting the results as expected. Notice that the address of c is 0x200ac, which is also 4-byte different with the address of b. This is also because b is a 4-byte word (Figure 9).


```

nbminh — pi@raspberrypi: ~/Project — ssh pi@192.168.43.160 — 80x66
13      add r1, r1, r2 @ add r1 to r2 and store into r1
(gdb) info registers
r0      0x0          0
r1      0x2          2
r2      0x5          5
r3      0x0          0
r4      0x0          0
r5      0x0          0
r6      0x0          0
r7      0x0          0
r8      0x0          0
r9      0x0          0
r10     0x0          0
r11     0x0          0
r12     0x0          0
sp      0x7efff650   0x7efff650
lr      0x0          0
pc      0x10084      0x10084 <_start+16>
cpsr    0x10        16
fpscr   0x0          0
(gdb) stepi
14      ldr r2, =c    @ load the memory address of c into r2
(gdb) info registers
r0      0x0          0
r1      0x7          7
r2      0x5          5
r3      0x0          0
r4      0x0          0
r5      0x0          0
r6      0x0          0
r7      0x0          0
r8      0x0          0
r9      0x0          0
r10     0x0          0
r11     0x0          0
r12     0x0          0
sp      0x7efff650   0x7efff650
lr      0x0          0
pc      0x10088      0x10088 <_start+20>
cpsr    0x10        16
fpscr   0x0          0
(gdb) stepi
15      str r1, [r2]  @ store r1 into memory c
(gdb) info registers
r0      0x0          0
r1      0x7          7
r2      0x200ac      131244
r3      0x0          0
r4      0x0          0
r5      0x0          0
r6      0x0          0
r7      0x0          0
r8      0x0          0
r9      0x0          0
r10     0x0          0
r11     0x0          0
r12     0x0          0
sp      0x7efff650   0x7efff650
lr      0x0          0
pc      0x1008c      0x1008c <_start+24>
cpsr    0x10        16
fpscr   0x0          0
(gdb) x/1xw 0x200ac
0x200ac: 0x00000000
(gdb) stepi
17      mov r7, #1    @ Program Termination: exit syscall

```

Figure 9

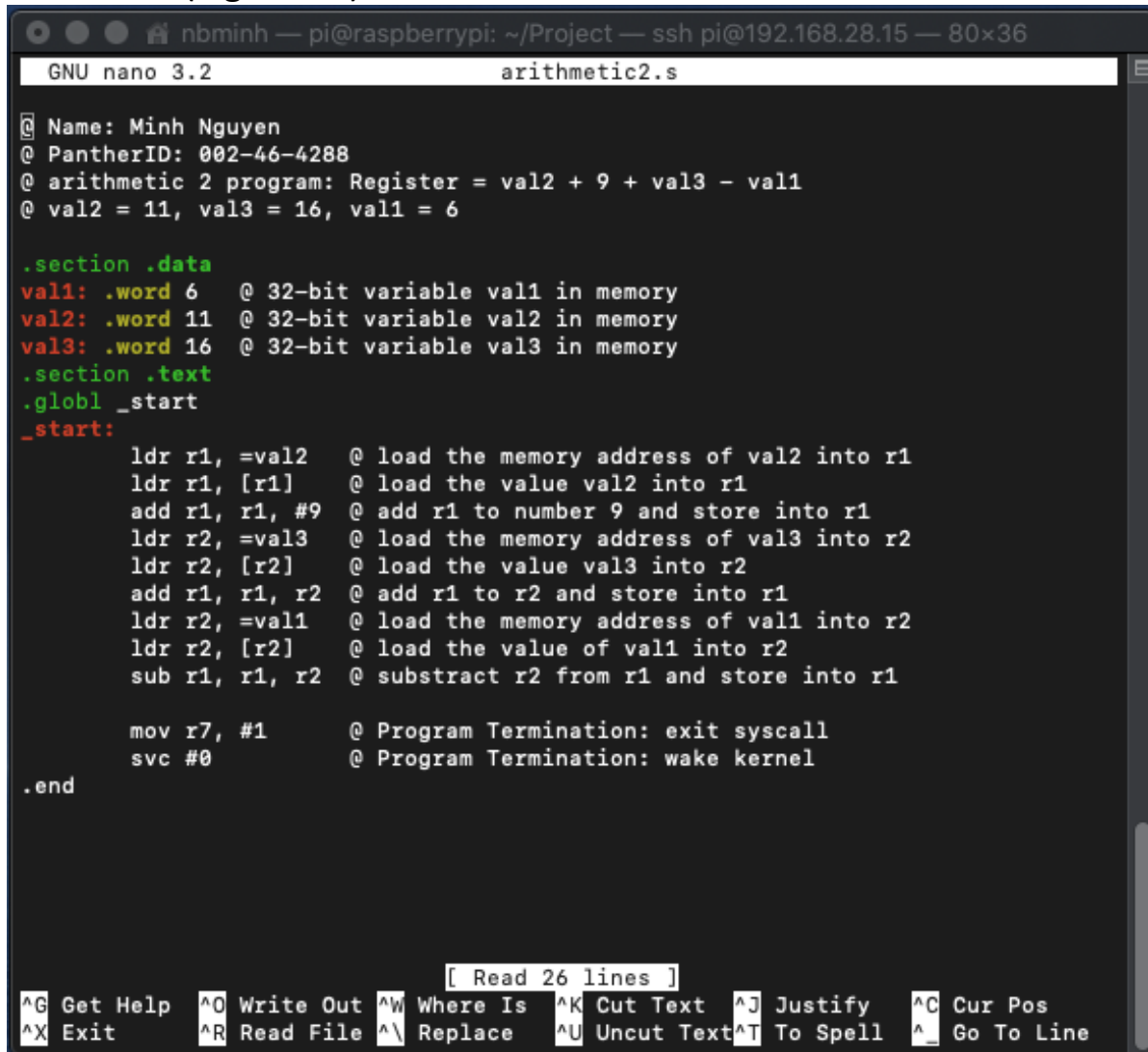
Before terminating the program, I went ahead and checked the memory at c's address. It stored the value 7, which is the correct result of the addition (Figure 10).

```
nbminh — pi@raspberrypi: ~/Project — -bash — 80x22
17      mov r7, #1      @ Program Termination: exit syscall
(gdb) info registers
r0      0x0             0
r1      0x7             7
r2      0x200ac         131244
r3      0x0             0
r4      0x0             0
r5      0x0             0
r6      0x0             0
r7      0x0             0
r8      0x0             0
r9      0x0             0
r10     0x0             0
r11     0x0             0
r12     0x0             0
sp      0x7efff650      0x7efff650
lr      0x0             0
pc      0x10090         0x10090 <_start+28>
cpsr    0x10           16
fpscr   0x0             0
(gdb) x/1xw 0x200ac
0x200ac: 0x00000007
```

Figure 10

Part 2

I created a arithmetic2.s file and type in the code for this program using the editor (Figure 11).



```
GNU nano 3.2 arithmetic2.s
@ Name: Minh Nguyen
@ PantherID: 002-46-4288
@ arithmetic 2 program: Register = val2 + 9 + val3 - val1
@ val2 = 11, val3 = 16, val1 = 6

.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, =val2 @ load the memory address of val2 into r1
    ldr r1, [r1] @ load the value val2 into r1
    add r1, r1, #9 @ add r1 to number 9 and store into r1
    ldr r2, =val3 @ load the memory address of val3 into r2
    ldr r2, [r2] @ load the value val3 into r2
    add r1, r1, r2 @ add r1 to r2 and store into r1
    ldr r2, =val1 @ load the memory address of val1 into r2
    ldr r2, [r2] @ load the value of val1 into r2
    sub r1, r1, r2 @ subtract r2 from r1 and store into r1

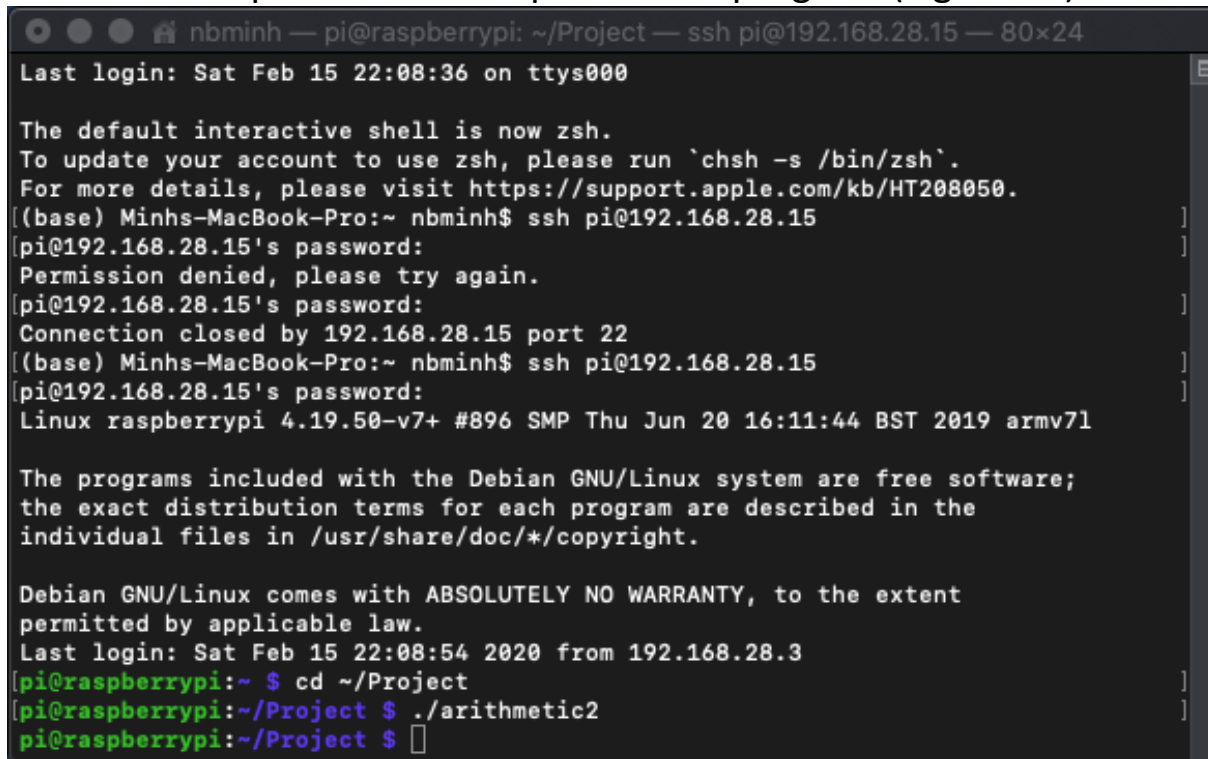
    mov r7, #1 @ Program Termination: exit syscall
    svc #0 @ Program Termination: wake kernel

.end

[ Read 26 lines ]
^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
```

Figure 11

Running it gives me no output just like part 1. Because there is no instruction to produce the output for this program (Figure 12).

A terminal window titled 'nbminh — pi@raspberrypi: ~/Project — ssh pi@192.168.28.15 — 80x24'. The terminal shows a successful SSH login to a Raspberry Pi. The user 'pi' provides a password, and the system responds with the default shell (zsh), system information (Linux raspberrypi 4.19.50-v7+), and the Debian GNU/Linux license. The user then navigates to the ~/Project directory and runs a script named ./arithmetic2, which produces no output.

```
nbminh — pi@raspberrypi: ~/Project — ssh pi@192.168.28.15 — 80x24
Last login: Sat Feb 15 22:08:36 on ttys000

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
[(base) Minhs-MacBook-Pro:~ nbminh$ ssh pi@192.168.28.15
pi@192.168.28.15's password:
Permission denied, please try again.
pi@192.168.28.15's password:
Connection closed by 192.168.28.15 port 22
[(base) Minhs-MacBook-Pro:~ nbminh$ ssh pi@192.168.28.15
pi@192.168.28.15's password:
Linux raspberrypi 4.19.50-v7+ #896 SMP Thu Jun 20 16:11:44 BST 2019 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sat Feb 15 22:08:54 2020 from 192.168.28.3
pi@raspberrypi:~ $ cd ~/Project
pi@raspberrypi:~/Project $ ./arithmetic2
pi@raspberrypi:~/Project $
```

Figure 12

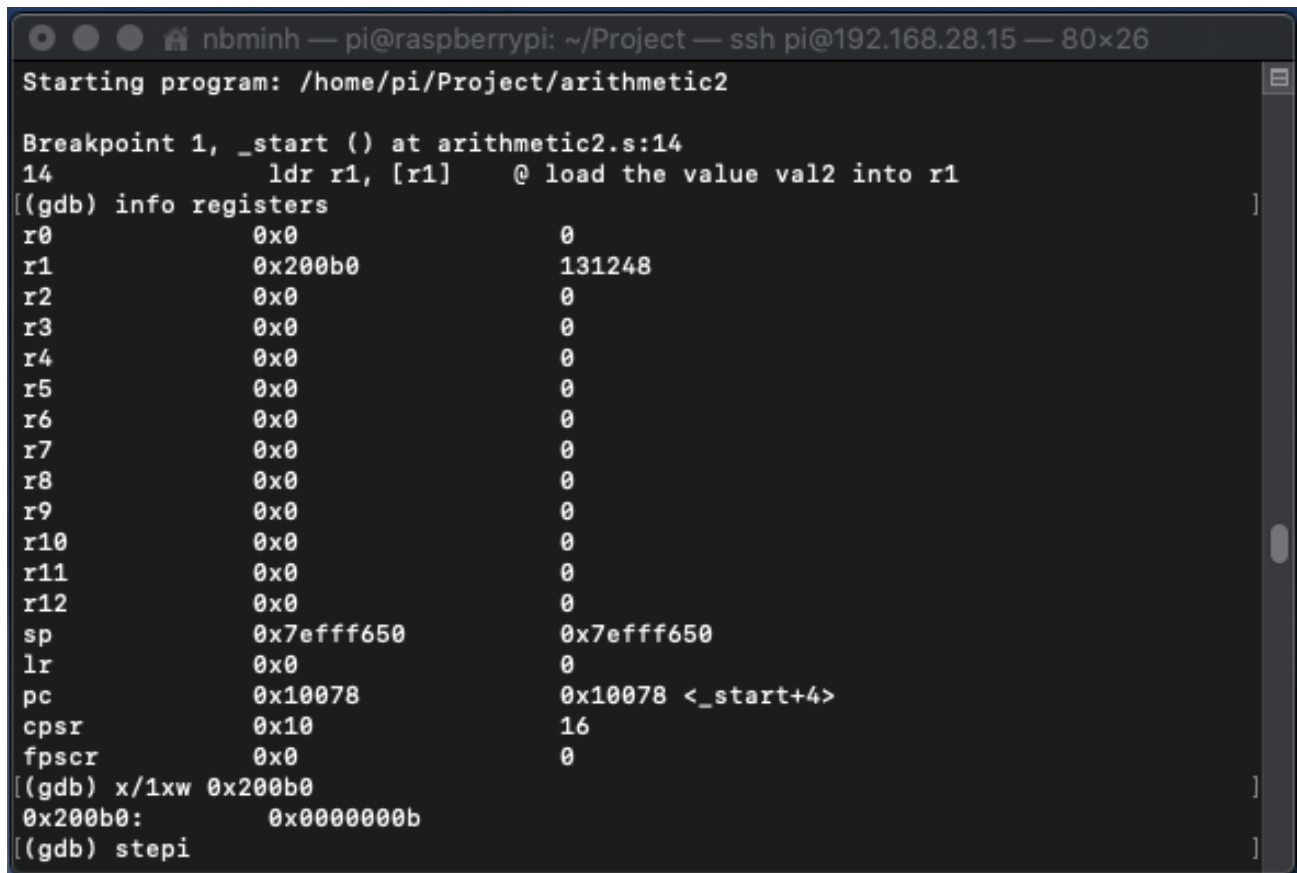
I started debugging it using GDB and set the breakpoint at line 14 (Figure 13).

```
nbminh — pi@raspberrypi: ~/Project — ssh pi@192.168.28.15 — 80x50
[pi@raspberrypi:~/Project $ 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-gnueabi".
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      @ Name: Minh Nguyen
2      @ PantherID: 002-46-4288
3      @ arithmetic 2 program: Register = val2 + 9 + val3 - val1
4      @ val2 = 11, val3 = 16, val1 = 6
5
6      .section .data
7      val1: .word 6      @ 32-bit variable val1 in memory
8      val2: .word 11     @ 32-bit variable val2 in memory
9      val3: .word 16     @ 32-bit variable val3 in memory
10     .section .text
((gdb)
11     .globl _start
12     _start:
13         ldr r1, =val2    @ load the memory address of val2 into r1
14         ldr r1, [r1]     @ load the value val2 into r1
15         add r1, r1, #9    @ add r1 to number 9 and store into r1
16         ldr r2, =val3    @ load the memory address of val3 into r2
17         ldr r2, [r2]     @ load the value val3 into r2
18         add r1, r1, r2    @ add r1 to r2 and store into r1
19         ldr r2, =val1    @ load the memory address of val1 into r2
20         ldr r2, [r2]     @ load the value of val1 into r2
((gdb)
21         sub r1, r1, r2    @ subtract r2 from r1 and store into r1
22
23         mov r7, #1        @ Program Termination: exit syscall
24         svc #0            @ Program Termination: wake kernel
25     .end
26
((gdb) b 13
Breakpoint 1 at 0x10078: file arithmetic2.s, line 14.
((gdb) run
Starting program: /home/pi/Project/arithmetic2
```

Figure 13

By using the same commands as part 1, I can step through the program line by line to examine the registers and memory data. After executing line 13, I got the address of val2 which is 0x200b0. Examining this address gave me the value b in hex (which is 11 in decimal) as expected (Figure 14).

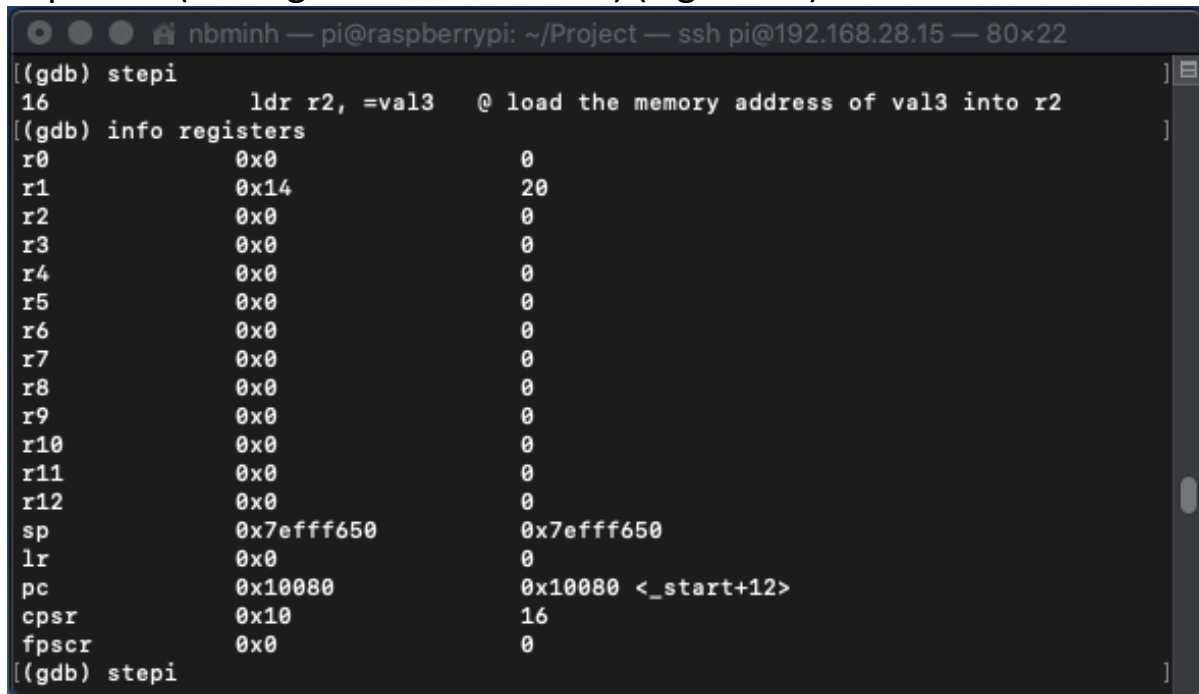


```
nbminh — pi@raspberrypi: ~/Project — ssh pi@192.168.28.15 — 80x26
Starting program: /home/pi/Project/arithmetic2

Breakpoint 1, _start () at arithmetic2.s:14
14      ldr r1, [r1]    @ load the value val2 into r1
[(gdb) info registers
r0             0x0             0
r1             0x200b0         131248
r2             0x0             0
r3             0x0             0
r4             0x0             0
r5             0x0             0
r6             0x0             0
r7             0x0             0
r8             0x0             0
r9             0x0             0
r10            0x0             0
r11            0x0             0
r12            0x0             0
sp             0x7efff650      0x7efff650
lr             0x0             0
pc             0x10078         0x10078 <_start+4>
cpsr           0x10           16
fpscr          0x0             0
[(gdb) x/1xw 0x200b0
0x200b0:      0x0000000b
[(gdb) stepi
```

Figure 14

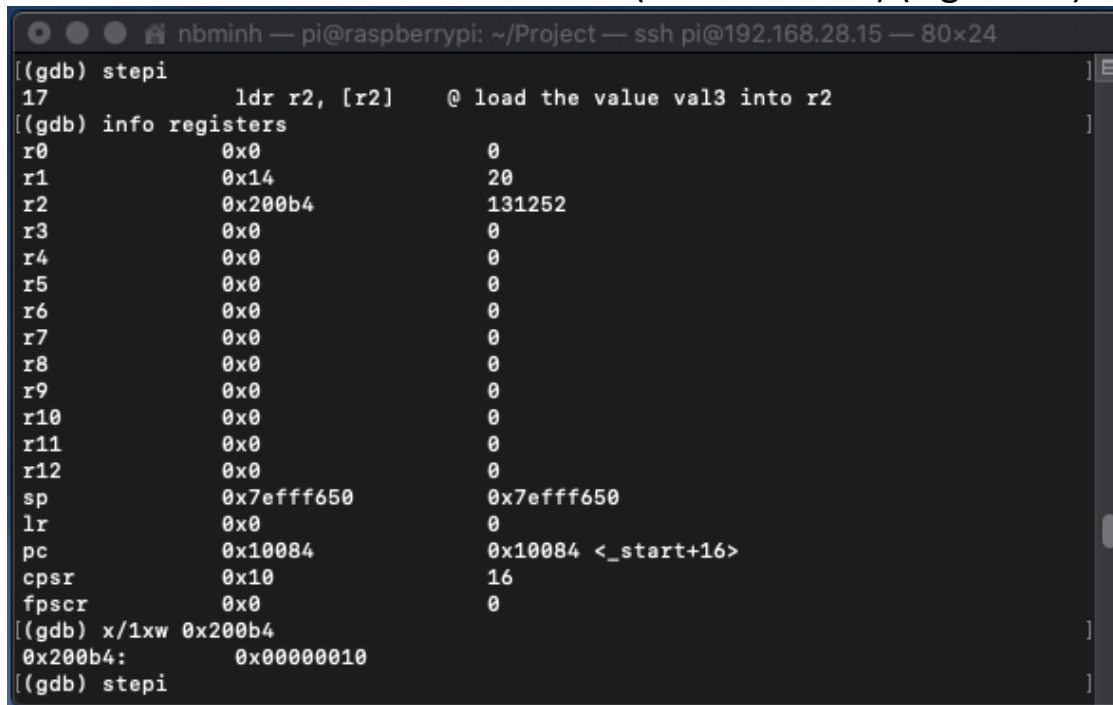
After executing line 15, I got the value 20 stored in r1 register as expected (adding val2 to number 9) (Figure 15).



```
nbminh — pi@raspberrypi: ~/Project — ssh pi@192.168.28.15 — 80x22
[(gdb) stepi
16          ldr r2, =val3    @ load the memory address of val3 into r2
[(gdb) info registers
r0          0x0             0
r1          0x14            20
r2          0x0             0
r3          0x0             0
r4          0x0             0
r5          0x0             0
r6          0x0             0
r7          0x0             0
r8          0x0             0
r9          0x0             0
r10         0x0             0
r11         0x0             0
r12         0x0             0
sp          0x7efff650      0x7efff650
lr          0x0             0
pc          0x10080         0x10080 <_start+12>
cpsr       0x10            16
fpscr      0x0             0
[(gdb) stepi
```

Figure 15

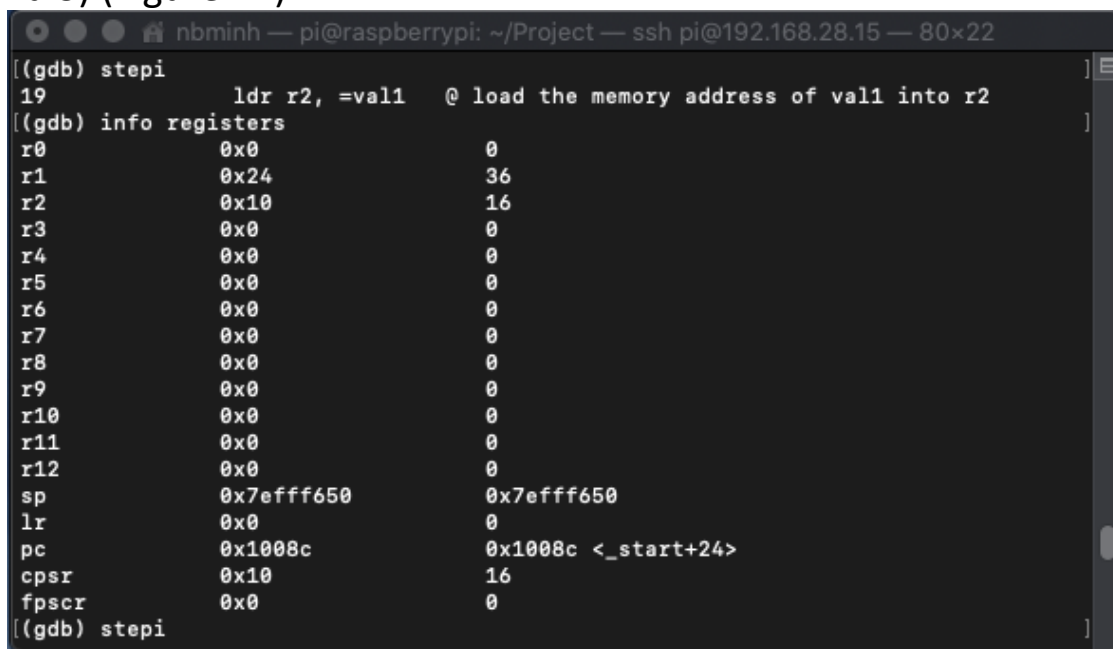
Executing line 16 gave me the address of val3, which is 0x200b4. The value this address holds is 10 in hex (16 in decimal) (Figure 16).



```
nbminh — pi@raspberrypi: ~/Project — ssh pi@192.168.28.15 — 80x24
[(gdb) stepi
17      ldr r2, [r2]    @ load the value val3 into r2
[(gdb) info registers
r0          0x0         0
r1          0x14        20
r2          0x200b4     131252
r3          0x0         0
r4          0x0         0
r5          0x0         0
r6          0x0         0
r7          0x0         0
r8          0x0         0
r9          0x0         0
r10         0x0         0
r11         0x0         0
r12         0x0         0
sp          0x7efff650   0x7efff650
lr          0x0         0
pc          0x10084     0x10084 <_start+16>
cpsr       0x10        16
fpscr      0x0         0
[(gdb) x/1xw 0x200b4
0x200b4:    0x00000010
[(gdb) stepi
```

Figure 16

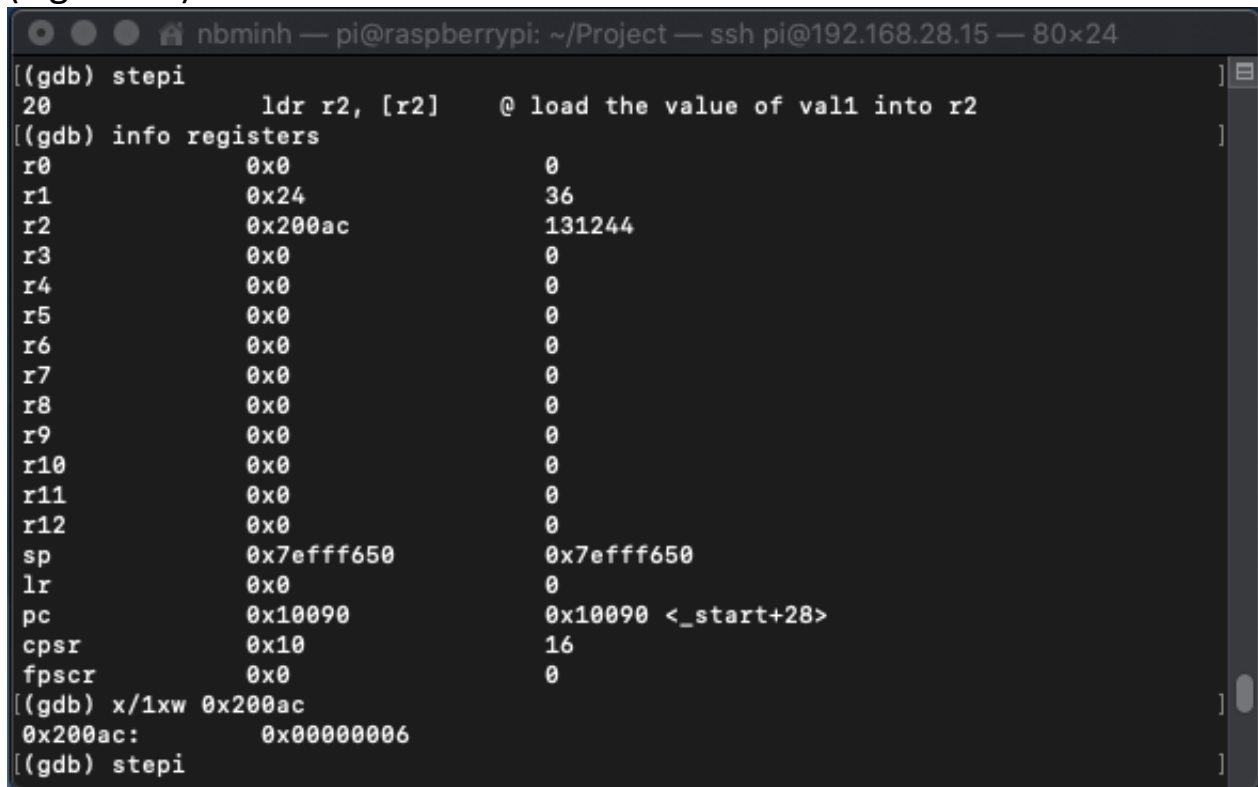
After the addition in line 18, r1 register holds the value 36 (val2 + 9 + val3) (Figure 17).



```
nbminh — pi@raspberrypi: ~/Project — ssh pi@192.168.28.15 — 80x22
[(gdb) stepi
19      ldr r2, =val1   @ load the memory address of val1 into r2
[(gdb) info registers
r0          0x0         0
r1          0x24        36
r2          0x10        16
r3          0x0         0
r4          0x0         0
r5          0x0         0
r6          0x0         0
r7          0x0         0
r8          0x0         0
r9          0x0         0
r10         0x0         0
r11         0x0         0
r12         0x0         0
sp          0x7efff650   0x7efff650
lr          0x0         0
pc          0x1008c     0x1008c <_start+24>
cpsr       0x10        16
fpscr      0x0         0
[(gdb) stepi
```

Figure 17

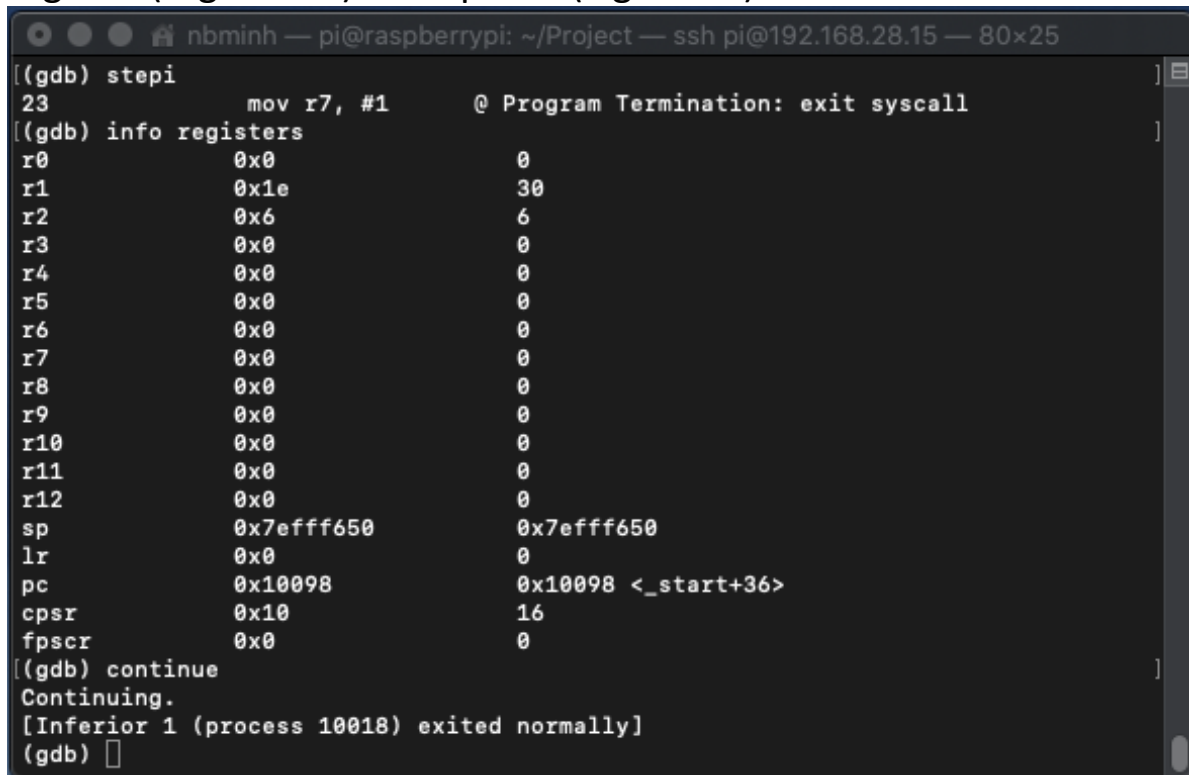
After executing line 19, I had the address of val1 in r2 which is 0x200ac. Reading from this address gave me the value 6 in hex (also 6 in decimal) (Figure 18).

A screenshot of a GDB terminal window. The window title is 'nbminh — pi@raspberrypi: ~/Project — ssh pi@192.168.28.15 — 80x24'. The terminal shows the following commands and output:

```
[(gdb) stepi
20          ldr r2, [r2]      @ load the value of val1 into r2
[(gdb) info registers
r0          0x0              0
r1          0x24             36
r2          0x200ac          131244
r3          0x0              0
r4          0x0              0
r5          0x0              0
r6          0x0              0
r7          0x0              0
r8          0x0              0
r9          0x0              0
r10         0x0              0
r11         0x0              0
r12         0x0              0
sp          0x7efff650       0x7efff650
lr          0x0              0
pc          0x10090          0x10090 <_start+28>
cpsr        0x10             16
fpscr       0x0              0
[(gdb) x/1xw 0x200ac
0x200ac:    0x00000006
[(gdb) stepi
```

Figure 18

The result of the arithmetic expression is 30 and it was stored in a register (register r1) as required (Figure 19).



```
nbminh — pi@raspberrypi: ~/Project — ssh pi@192.168.28.15 — 80x25
[(gdb) stepi
23          mov r7, #1      @ Program Termination: exit syscall
[(gdb) info registers
r0          0x0             0
r1          0x1e            30
r2          0x6             6
r3          0x0             0
r4          0x0             0
r5          0x0             0
r6          0x0             0
r7          0x0             0
r8          0x0             0
r9          0x0             0
r10         0x0             0
r11         0x0             0
r12         0x0             0
sp          0x7efff650      0x7efff650
lr          0x0             0
pc          0x10098         0x10098 <_start+36>
cpsr        0x10            16
fpscr       0x0             0
[(gdb) continue
Continuing.
[Inferior 1 (process 10018) exited normally]
(gdb) 
```

Figure 19

And here is what my Project folder in Raspberry looks like. Noted that the selected files are all the files that were created in this project (Figure 20).

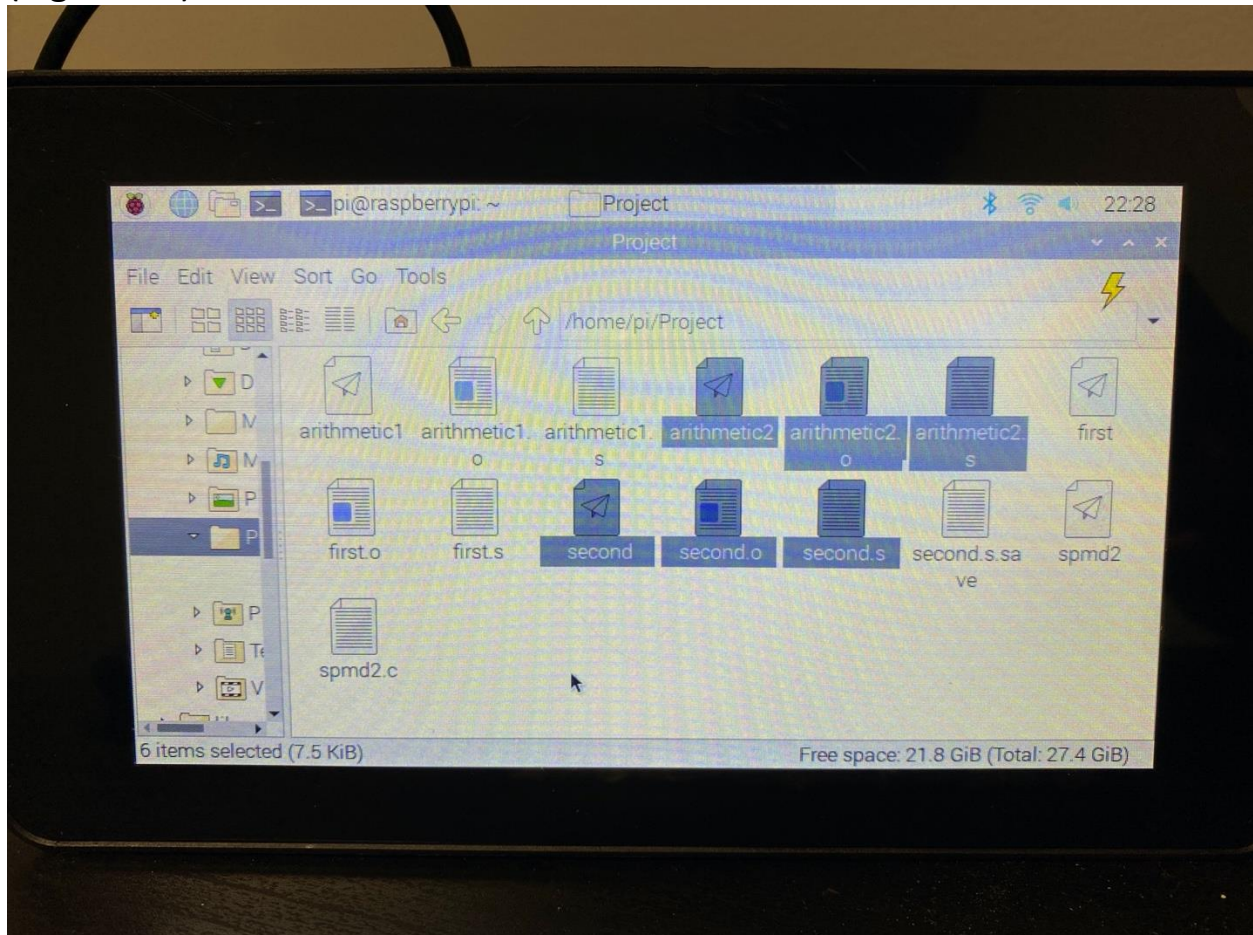


Figure 20