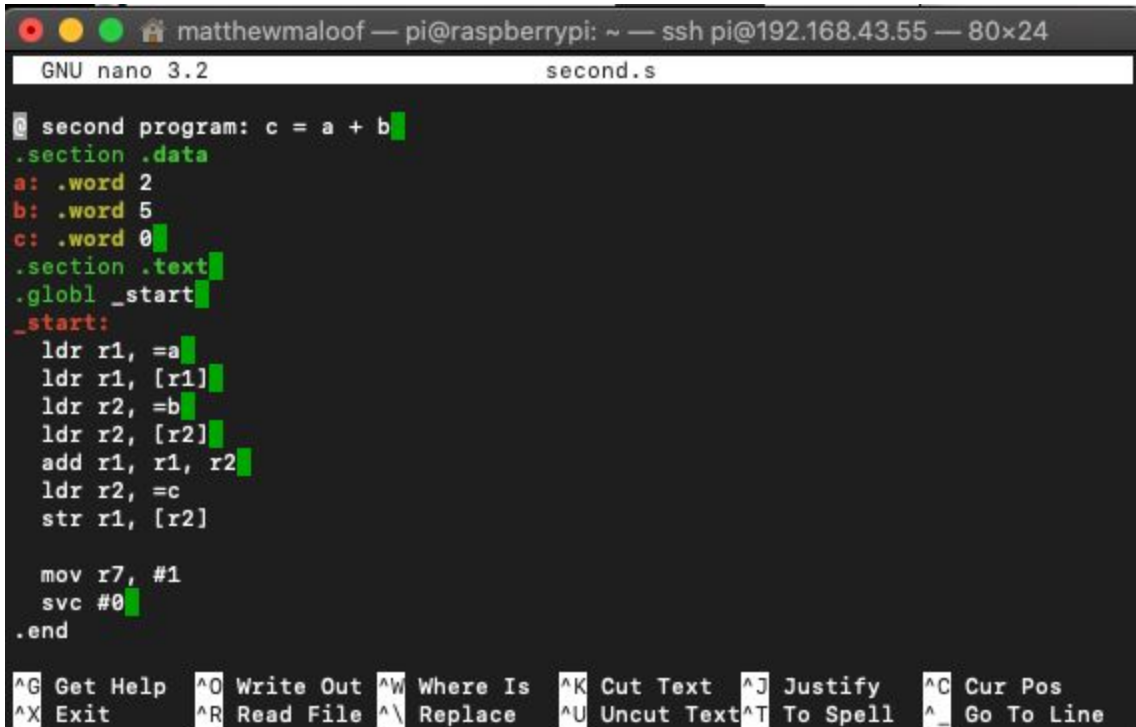


Matthew Maloof

Task 4: ARM Assembly Programming

First step is to go into the terminal and type nano second.s to create a file named second.s and edit the contents. Figure 1 shows the contents of this file. Then you press control+x, y, then enter to save the file and exit the editor.



```
GNU nano 3.2 second.s

second program: c = a + b
.section .data
a: .word 2
b: .word 5
c: .word 0
.section .text
.globl _start
_start:
    ldr r1, =a
    ldr r1, [r1]
    ldr r2, =b
    ldr r2, [r2]
    add r1, r1, r2
    ldr r2, =c
    str r1, [r2]

    mov r7, #1
    svc #0
.end

^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 1: Code for second.s file

We then type `as -o second.o second.s` to assemble the file, `ld -o second second.o` to link the file and `./second` to run the file that is now an executable. As seen in figure 2 below, there is no output for this program which is expected because no output was set in the code, it is just running a program. It'd be the equivalent of having a calculator with no screen, it would still do the calculation but no output would be given/shown for the user.

```
matthewmaloof — pi@raspberrypi: ~ — ssh pi@192.168.43.55 — 80x24

[pi@raspberrypi:~ $ ld -o second second.o
[pi@raspberrypi:~ $ ./second
[pi@raspberrypi:~ $
```

Figure 2: Turn into executable no output

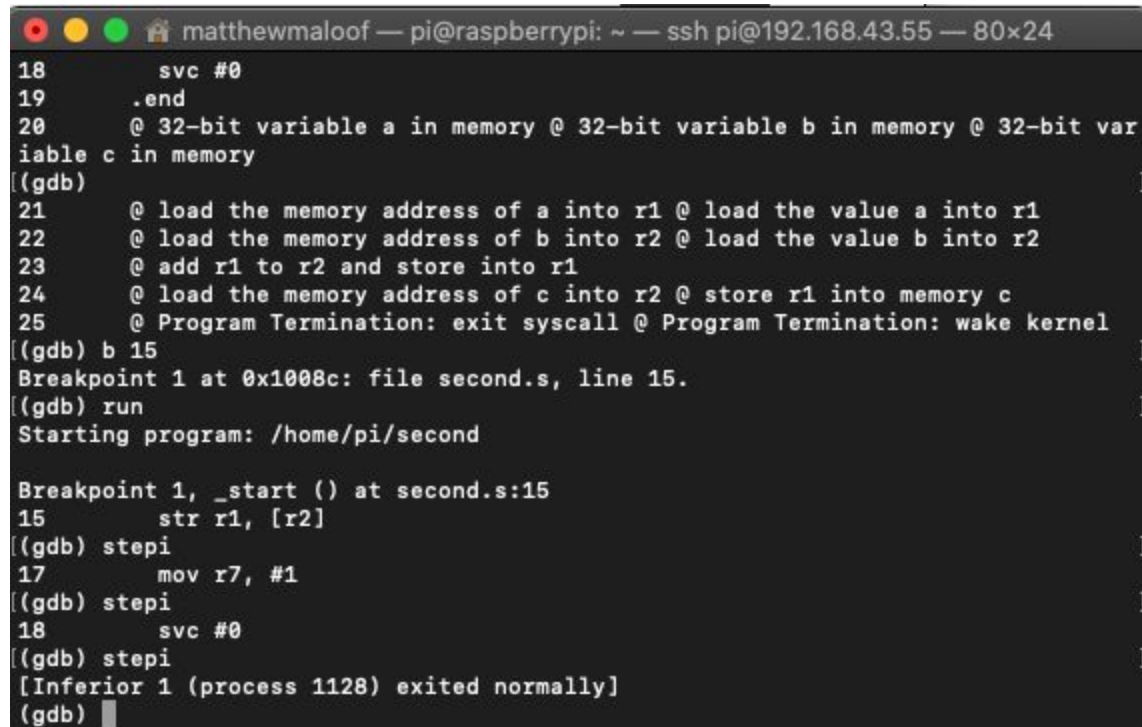
We then type `gdb second` to enter the GNU debugger.

```
matthewmaloof — pi@raspberrypi: ~ — ssh pi@192.168.43.55 — 80x24
-bash: gdb: command not found
[pi@raspberrypi:~ $ gdb second
GNU gdb (Raspbian 8.2.1-2) 8.2.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "arm-linux-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
4      b: .word 5
5      c: .word 0
```

Figure 3: GDB into GNU debugger

We then set a breakpoint at line 15 by typing `b 15` then type `run`.

A screenshot of a terminal window with a dark background and light text. The window title bar shows a Raspberry Pi icon, the username 'matthewmaloof', the host 'pi@raspberrypi', the directory '~', and the connection 'ssh pi@192.168.43.55' with a resolution of '80x24'. The terminal content shows assembly code for lines 18 through 25, followed by GDB commands and their output. The commands include setting a breakpoint at line 15, running the program, and stepping through instructions. The program terminates normally.

```
18      svc #0
19      .end
20      @ 32-bit variable a in memory @ 32-bit variable b in memory @ 32-bit var
variable c in memory
(gdb)
21      @ load the memory address of a into r1 @ load the value a into r1
22      @ load the memory address of b into r2 @ load the value b into r2
23      @ add r1 to r2 and store into r1
24      @ load the memory address of c into r2 @ store r1 into memory c
25      @ Program Termination: exit syscall @ Program Termination: wake kernel
(gdb) b 15
Breakpoint 1 at 0x1008c: file second.s, line 15.
(gdb) run
Starting program: /home/pi/second

Breakpoint 1, _start () at second.s:15
15      str r1, [r2]
(gdb) stepi
17      mov r7, #1
(gdb) stepi
18      svc #0
(gdb) stepi
[Inferior 1 (process 1128) exited normally]
(gdb)
```

Figure 4: Breakpoint + run debugging

We type `stepi` to step through the instructions one at a time to see what the program is doing.

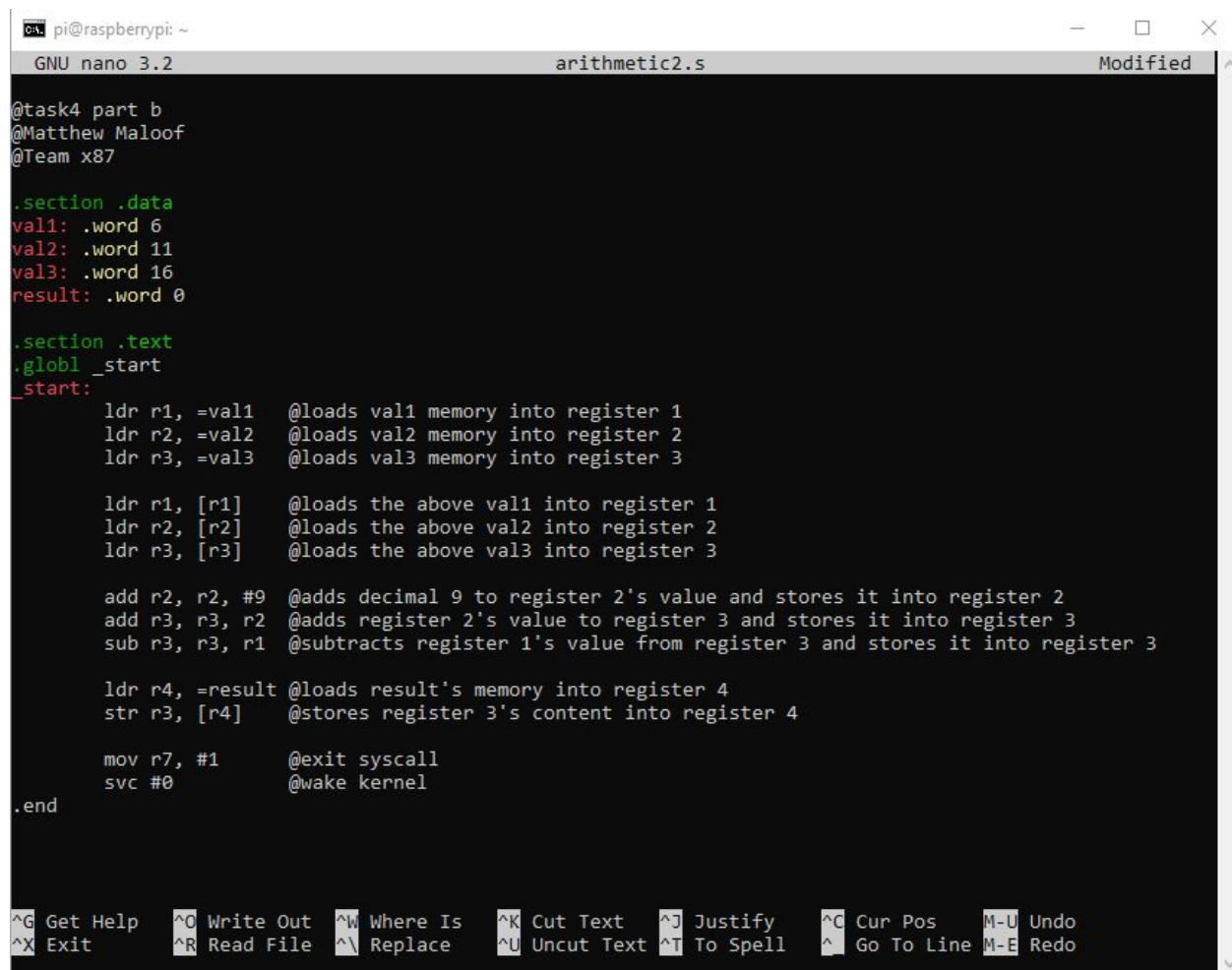
```
matthewmaloof — pi@raspberrypi: ~ — ssh pi@192.168.43.55 — 80x51
(gdb) info registers
r0          0x0          0
r1          0x200a4       131236
r2          0x0          0
r3          0x0          0
r4          0x0          0
r5          0x0          0
r6          0x0          0
r7          0x0          0
r8          0x0          0
r9          0x0          0
r10         0x0          0
r11         0x0          0
r12         0x0          0
sp          0x7efff680    0x7efff680
lr          0x0          0
pc          0x10078       0x10078 <_start+4>
cpsr        0x10         16
fpscr       0x0          0
(gdb) x/3xw 0x200a4
0x200a4:    0x00000002      0x00000005      0x00000000
(gdb) stepio
Undefined command: "stepio". Try "help".
(gdb) stepi
11      ldr r2, =b
(gdb) stepi
12      ldr r2, [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          0x7efff680    0x7efff680
lr          0x0          0
pc          0x10080       0x10080 <_start+12>
cpsr        0x10         16
fpscr       0x0          0
(gdb) x/3xw 0x200a8
0x200a8:    0x00000005      0x00000000      0x00001141
(gdb) x/1xw 0x200a8
0x200a8:    0x00000005
(gdb)
```

Figure 5: Info registers and stepi and x/1xw 0x200a8 command

In figure 5, we can see the contents of the registers by typing `stepi` then `info registers`. Register 2 has the value it should with `0x200a8` and `131240` in the right column. To display in hexadecimal, we can use the command `x/1xw` to display one hexadecimal followed by the address in the register, which in this case is `0x200a8`, so `x/1xw 0x200a8`. The output of this is shown above which is `0x00000005` which is correct.

Part B:

Using the template from Part A above, we create the code for `arithmetic2.s` with the instructions given in the slides.



```
pi@raspberrypi: ~
GNU nano 3.2 arithmetic2.s Modified
@task4 part b
@Matthew Maloof
@Team x87

.section .data
val1: .word 6
val2: .word 11
val3: .word 16
result: .word 0

.section .text
.globl _start
_start:
    ldr r1, =val1    @loads val1 memory into register 1
    ldr r2, =val2    @loads val2 memory into register 2
    ldr r3, =val3    @loads val3 memory into register 3

    ldr r1, [r1]      @loads the above val1 into register 1
    ldr r2, [r2]      @loads the above val2 into register 2
    ldr r3, [r3]      @loads the above val3 into register 3

    add r2, r2, #9    @adds decimal 9 to register 2's value and stores it into register 2
    add r3, r3, r2     @adds register 2's value to register 3 and stores it into register 3
    sub r3, r3, r1     @subtracts register 1's value from register 3 and stores it into register 3

    ldr r4, =result   @loads result's memory into register 4
    str r3, [r4]       @stores register 3's content into register 4

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

.end

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

Figure 6: `arithmetic2.s` code

We can then compile and turn the file into an executable. After running this code, no output is shown just like in Part A. Next, we are going to debug the program so we can see what's being run/stored in the registers.

```
pi@raspberrypi: ~  
pi@raspberrypi:~$ as -o arithmetic2.o arithmetic2.s  
pi@raspberrypi:~$ ld -o arithmetic2 arithmetic2.o  
pi@raspberrypi:~$ ./arithmetic2  
pi@raspberrypi:~$
```

Figure 7: arithmetic2 code not showing any output

```
27          str r3, [r4]    @stores register 3's content into register 4  
(gdb) info registers  
r0           0x0           0  
r1           0x6           6  
r2           0x14          20  
r3           0x1e          30  
r4           0x200c4       131268  
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           0x7efff670    0x7efff670  
lr           0x0           0  
pc           0x1009c       0x1009c <_start+40>  
cpsr         0x10         16  
fpscr        0x0           0
```

Figure 8: debugging continued

We enter gdb GNU debugging mode and set a breakpoint at line 27, then use stepi and info registers to view what's in the registers at each line of code as we progress through the file shown in figure 8 (these are the final values in the registers) Finally, we use the command in Part A to view the memory address in hex. x/1xw 0x200c4 which gives the hex value 0x0000001e.