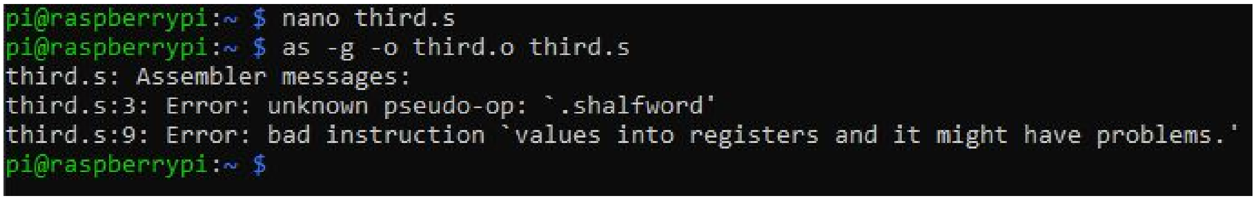
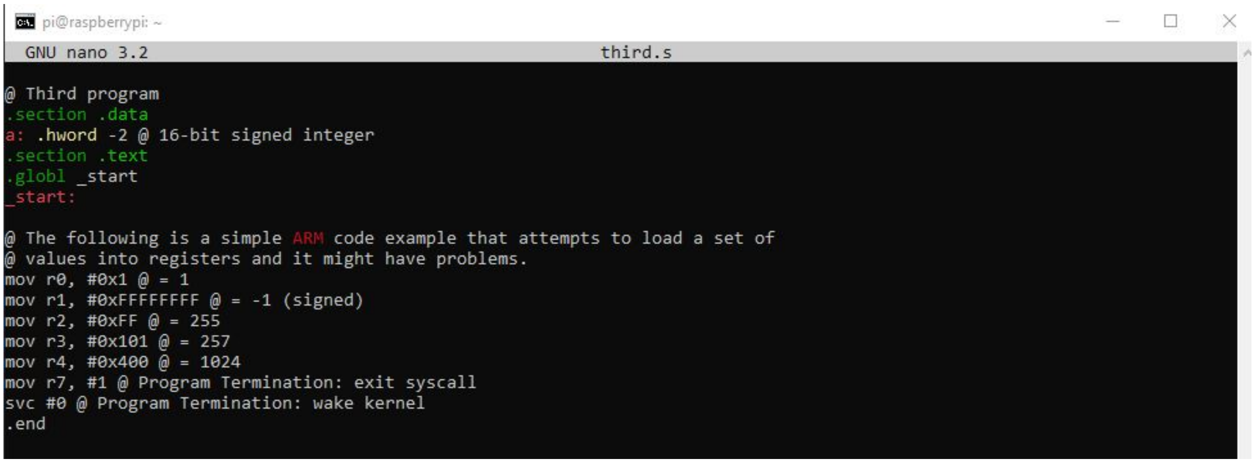
Matthew Maloof
Project 3 Task 4



```
pi@raspberrypi:~ $ nano third.s
pi@raspberrypi:~ $ as -g -o third.o third.s
third.s: Assembler messages:
third.s:3: Error: unknown pseudo-op: `.shalfword'
third.s:9: Error: bad instruction `values into registers and it might have problems.'
pi@raspberrypi:~ $
```

Figure 1

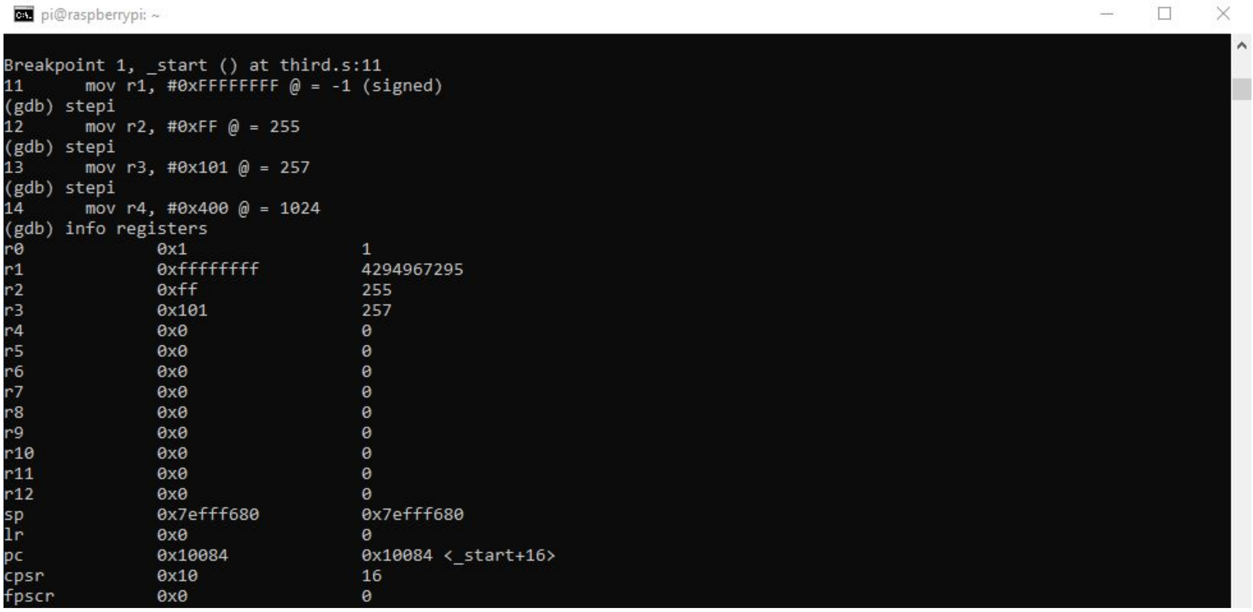
I created the file third.s using nano and copying and pasting the code given. The above is the error shown when you try to assemble the file using as command (unknown psuedo-op: '.shalfword')



```
@ Third program
.section .data
a: .hword -2 @ 16-bit signed integer
.section .text
.globl _start
_start:

@ The following is a simple ARM code example that attempts to load a set of
@ values into registers and it might have problems.
mov r0, #0x1 @ = 1
mov r1, #0xFFFFFFFF @ = -1 (signed)
mov r2, #0xFF @ = 255
mov r3, #0x101 @ = 257
mov r4, #0x400 @ = 1024
mov r7, #1 @ Program Termination: exit syscall
svc #0 @ Program Termination: wake kernel
.end
```

Figure 2

To fix this error, I changed the variable declaration a: .shalfword -2 to a: .hword -2, because using .shalfword is incorrect syntax under .data (Figure 2)
Then, we use the commands as -g -o third.o third.s AND ld -o third third.o to assemble and link the command respectively.

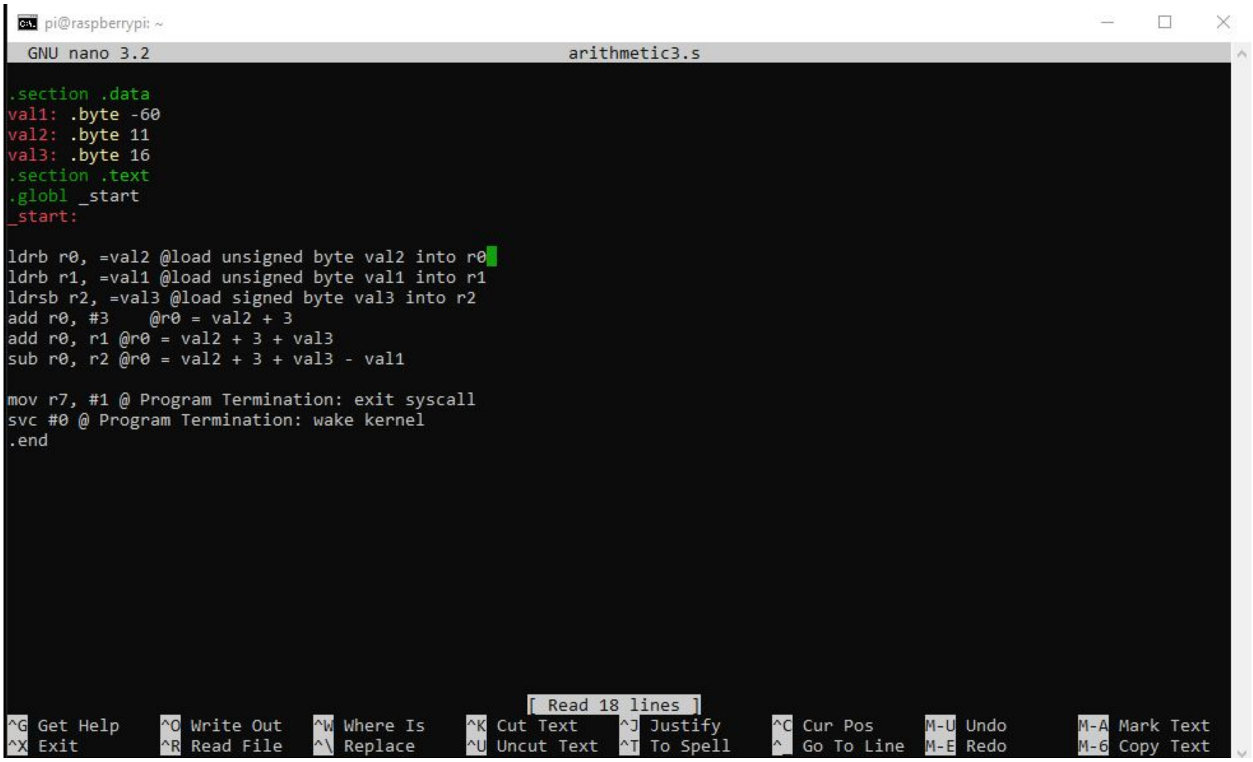


```
Breakpoint 1, _start () at third.s:11
11      mov r1, #0xFFFFFFFF @ = -1 (signed)
(gdb) stepi
12      mov r2, #0xFF @ = 255
(gdb) stepi
13      mov r3, #0x101 @ = 257
(gdb) stepi
14      mov r4, #0x400 @ = 1024
(gdb) info registers
r0             0x1                 1
r1             0xffffffff          4294967295
r2             0xff                255
r3             0x101               257
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             0x10084             0x10084 <_start+16>
cpsr           0x10                16
fpscr          0x0                 0
```

Figure 3

We then put a breakpoint at the start and go through the program using stepi. The registers information is shown above using info registers.

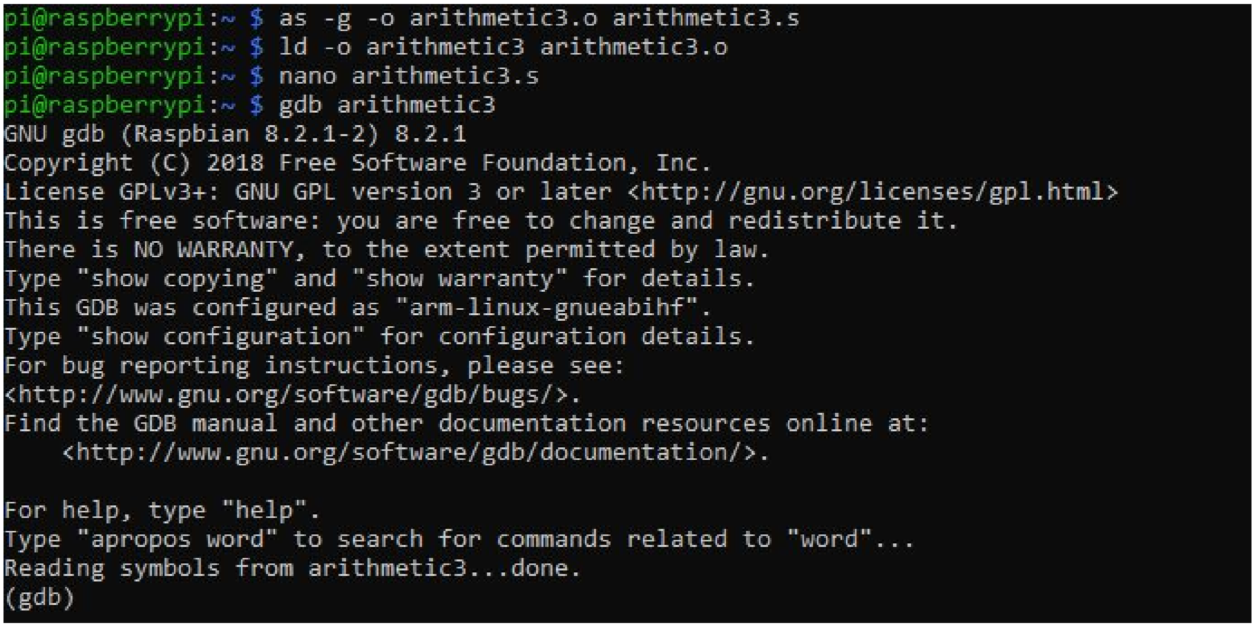
Then, we use the command x/1xh meaning hex and halfword display the memory address 1 item -- on the memory showing us the value inside. Then we use x/1xsh to show the signed memory.



```
  GNU nano 3.2                              arithmetic3.s

.section .data
val1: .byte -60
val2: .byte 11
val3: .byte 16
.section .text
.globl _start
_start:

ldrb r0, =val2 @load unsigned byte val2 into r0
ldrb r1, =val1 @load unsigned byte val1 into r1
ldrsb r2, =val3 @load signed byte val3 into r2
add r0, #3    @r0 = val2 + 3
add r0, r1 @r0 = val2 + 3 + val3
sub r0, r2 @r0 = val2 + 3 + val3 - val1

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

                                  [ Read 18 lines ]
^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
```

Figure 5

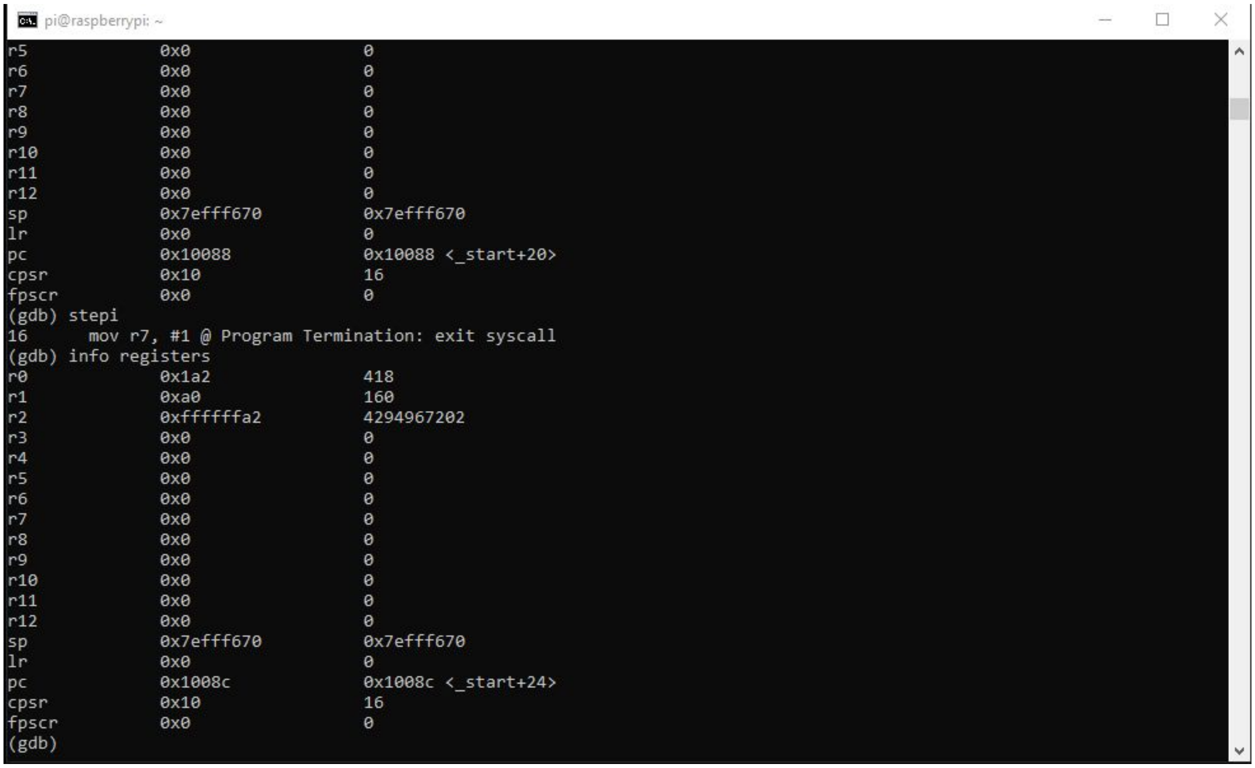
I created the file arithmetic3.s using nano command and inputted the commands shown in figure 5 for the equation given.



```
pi@raspberrypi:~ $ as -g -o arithmetic3.o arithmetic3.s
pi@raspberrypi:~ $ ld -o arithmetic3 arithmetic3.o
pi@raspberrypi:~ $ nano arithmetic3.s
pi@raspberrypi:~ $ gdb arithmetic3
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-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 arithmetic3...done.
(gdb)
```

Figure 6

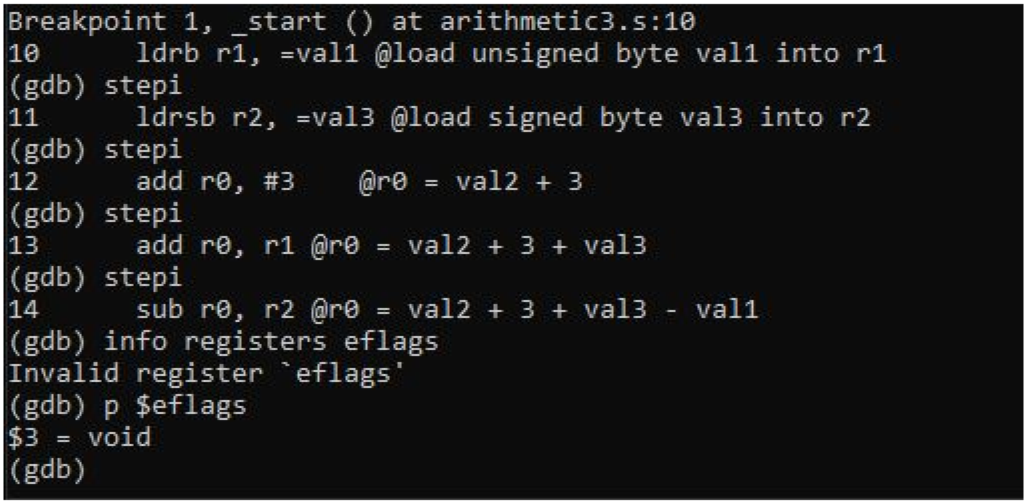
Then I assembled and linked the file and used gdb to enter debugging mode (Figure 6)



```
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             0x10088             0x10088 <_start+20>
cpsr           0x10                16
fpscr          0x0                 0
(gdb) stepi
16      mov r7, #1 @ Program Termination: exit syscall
(gdb) info registers
r0             0x1a2               418
r1             0xa0                160
r2             0xffffffa2          4294967202
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             0x7efff670          0x7efff670
lr             0x0                 0
pc             0x1008c             0x1008c <_start+24>
cpsr           0x10                16
fpscr          0x0                 0
(gdb)
```

Figure 7

I then set a breakpoint at line 10 using b 10 command and I typed run to run the program. I then typed stepi command to go through each line of code at a time and checked the registers using info registers.



```
Breakpoint 1, _start () at arithmetic3.s:10
10      ldrb r1, =val1 @load unsigned byte val1 into r1
(gdb) stepi
11      ldrsb r2, =val3 @load signed byte val3 into r2
(gdb) stepi
12      add r0, #3    @r0 = val2 + 3
(gdb) stepi
13      add r0, r1 @r0 = val2 + 3 + val3
(gdb) stepi
14      sub r0, r2 @r0 = val2 + 3 + val3 - val1
(gdb) info registers eflags
Invalid register `eflags'
(gdb) p $eflags
$3 = void
(gdb)
```

Figure 8

We then type p $eflags command to print out the flags, and as we can see in figure 8 there are no flags set. If I set val1 as the signed variable initially, and load the values into the registers using ldr r1, =val2 and ldrb r1, [r1] etc. , I would get 0x200ad as the memory value and using x/1xb 0x200ac would yield 0xc4. Then typing p/t $cpsr would yield an interrupt flag.