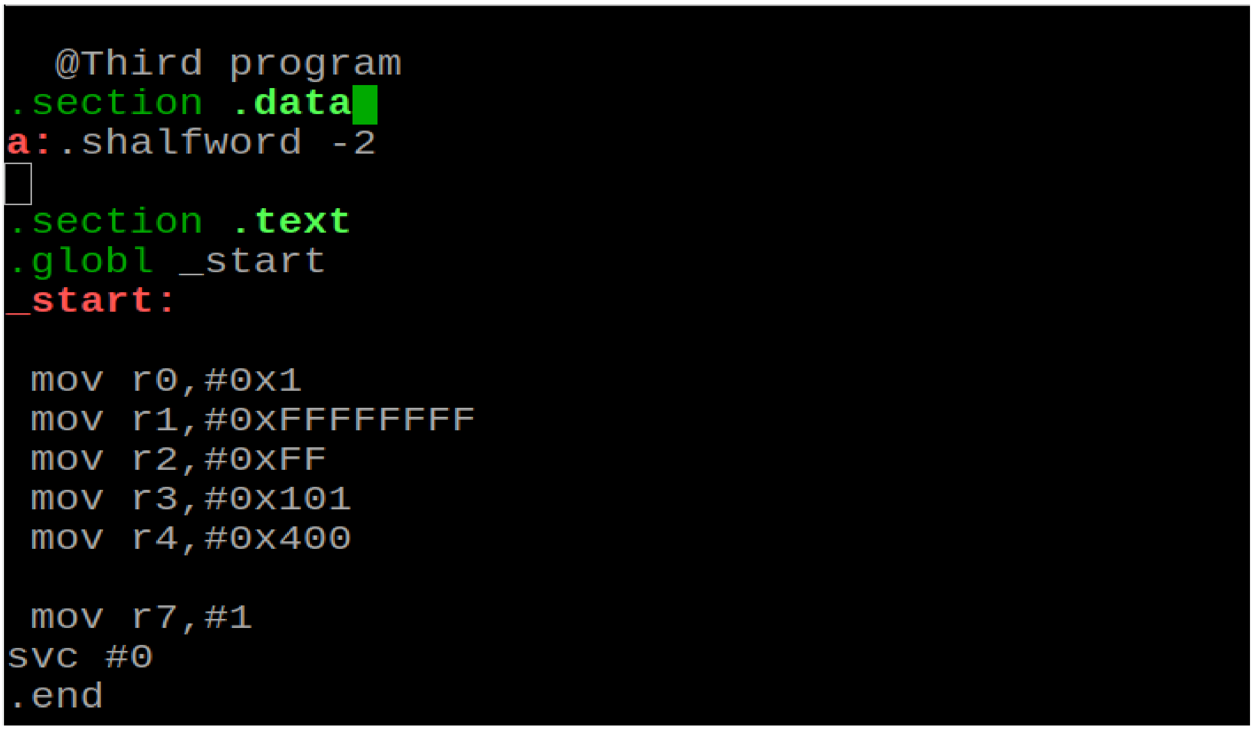
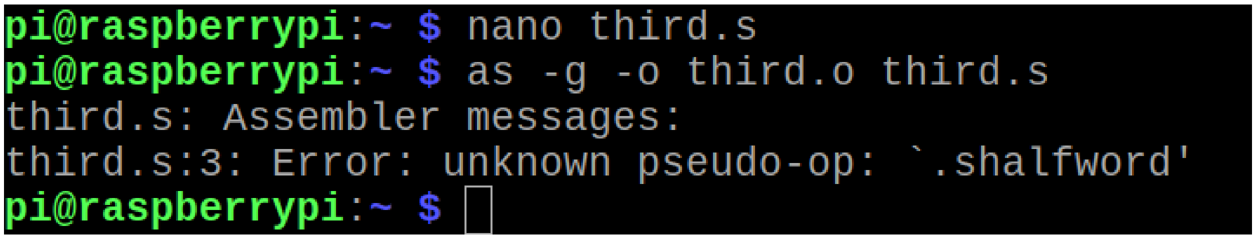
ARM Assembly Programming:

Part 1:
We started the assembly programming by opening a terminal, then opened a third.s file to write our program. We used the directive, .data, to indicate that we were going to declare some variables. We created a 2-byte signed memory size at location **a** initialized with -2. We then loaded the registers with some signed hexadecimal integers.



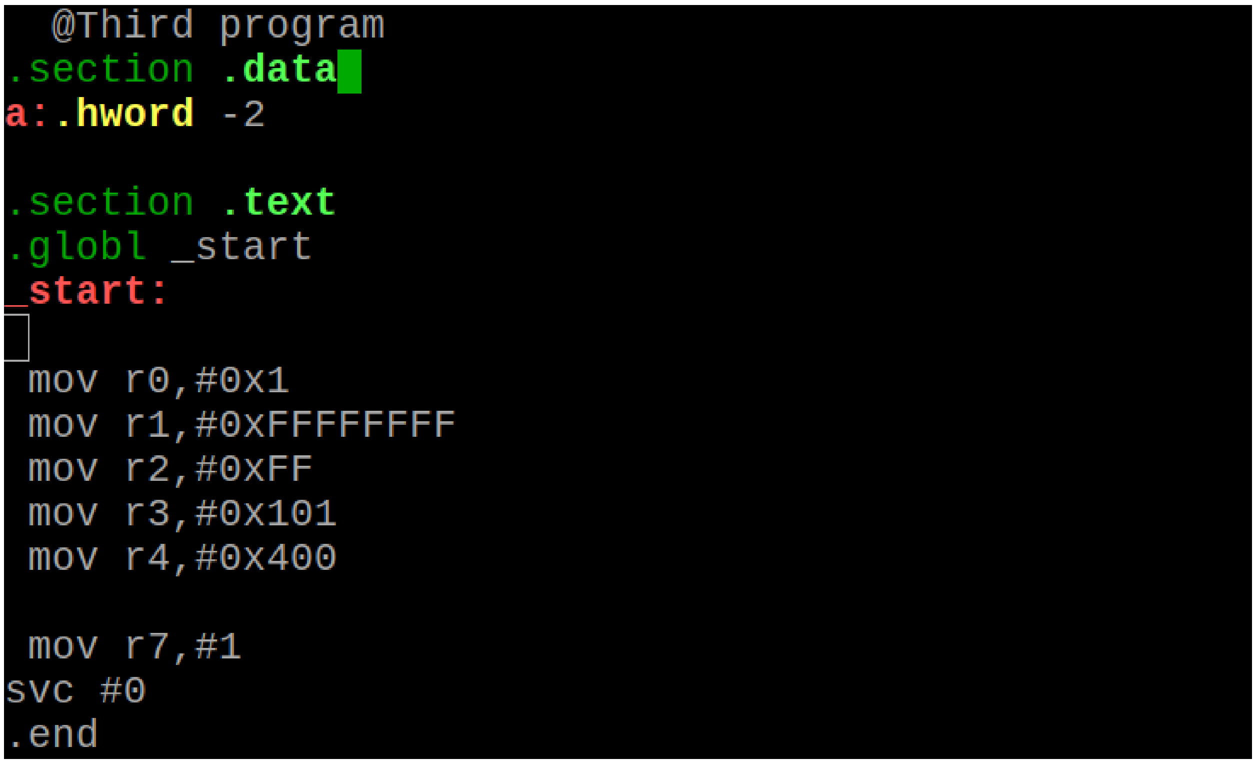
```
  @Third program
.section .data
a:.shalfword -2

.section .text
.globl _start
_start:

 mov r0,#0x1
 mov r1,#0xFFFFFFFF
 mov r2,#0xFF
 mov r3,#0x101
 mov r4,#0x400

 mov r7,#1
svc #0
.end
```

After we created the program, we tried to assemble it to get an object file and link, unfortunately, we couldn't assemble it because there was an error in the program with the following message:



```
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'
pi@raspberrypi:~ $
```

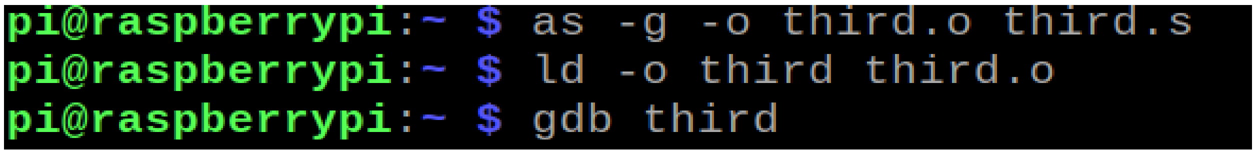
This error indicated that the 'shalfword' wasn't a keyword and therefore can't be used as a memory size. We fixed this error using the following program:



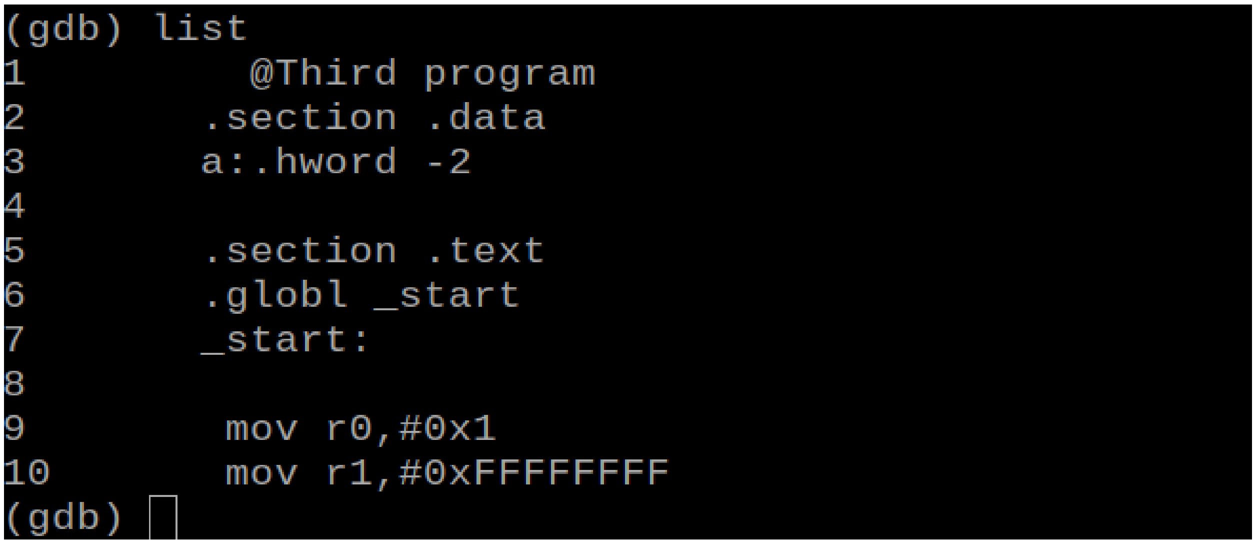
```
  @Third program
.section .data
a:.hword -2

.section .text
.globl _start
_start:

 mov r0,#0x1
 mov r1,#0xFFFFFFFF
 mov r2,#0xFF
 mov r3,#0x101
 mov r4,#0x400

 mov r7,#1
svc #0
.end
```

After the fix, we assembled and linked the program again, fortunately, it was able to assemble and link. We launched the gdb debugger for third.

We ran the **list** command to make sure our program was correct.

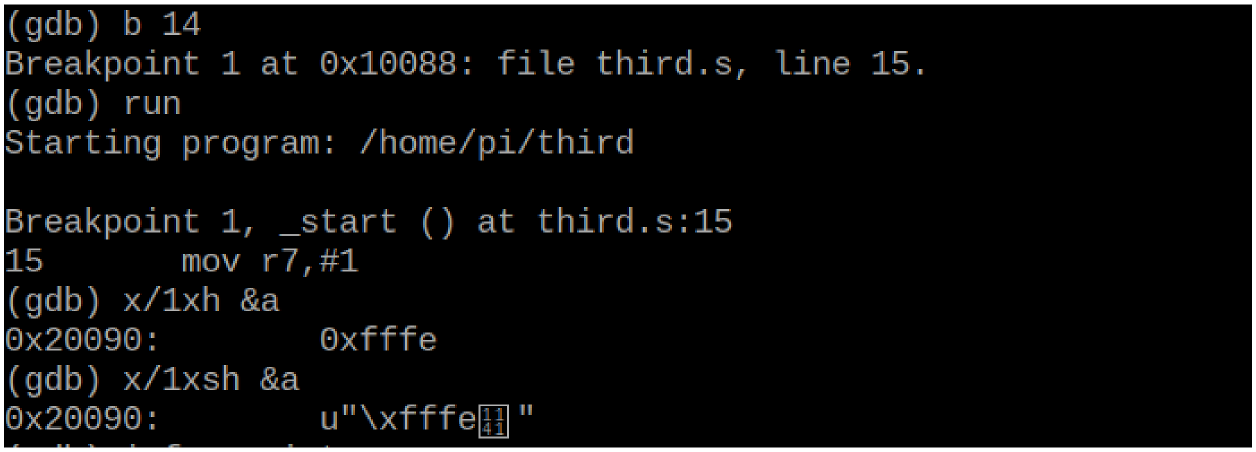


```
pi@raspberrypi:~ $ as -g -o third.o third.s
pi@raspberrypi:~ $ ld -o third third.o
pi@raspberrypi:~ $ gdb third
```



```
(gdb) list
1         @Third program
2       .section .data
3       a:.hword -2
4
5       .section .text
6       .globl _start
7       _start:
8
9        mov r0,#0x1
10       mov r1,#0xFFFFFFFF
(gdb)
```

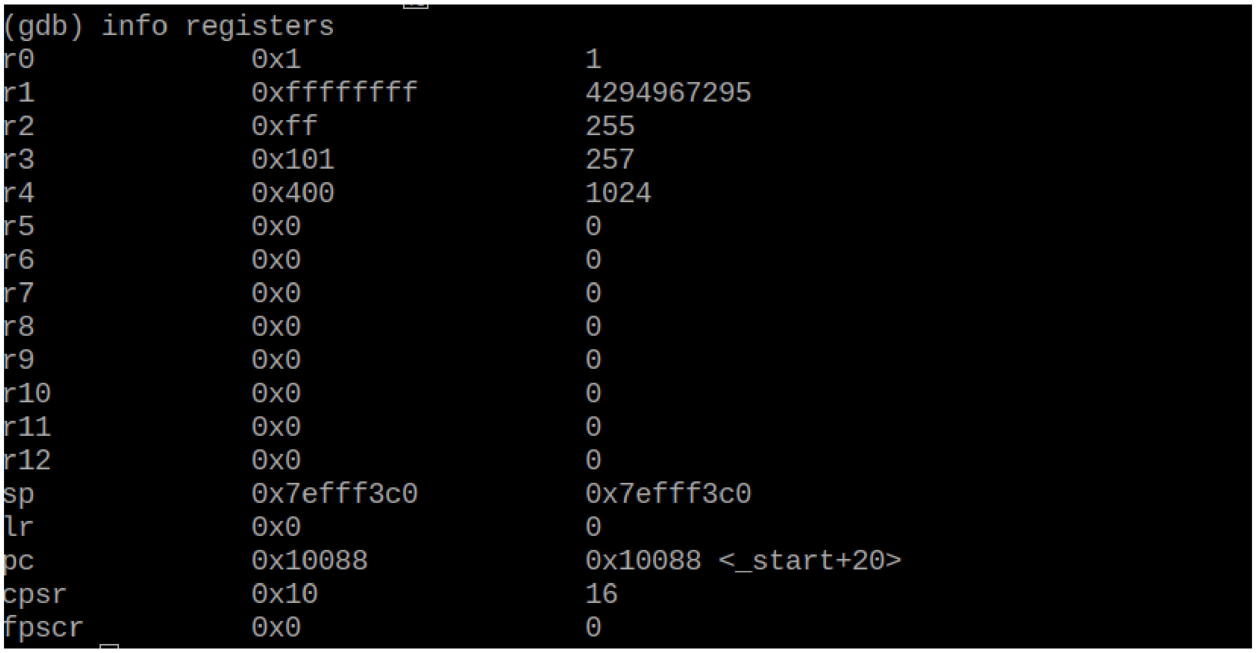
Since we didn't load the variable **a** into any register, it didn't matter where we set the breakpoint as long as it is outside the **.data section**. We set a breakpoint at line 14 using the syntex **b 14** to avoid stepping an instruction each time since we will be showing the content of the registers too. After that, we ran the program using **run**. After the program was successfully executed, we checked the memory location to make sure that the integer initialized (-2) was present in the 2-byte sized memory. We used both **x/1xh** and **x/1xsh** accompaning with the address of **a** (**&a)** to display the content of the memory in hexadecimal.



```
(gdb) b 14
Breakpoint 1 at 0x10088: file third.s, line 15.
(gdb) run
Starting program: /home/pi/third

Breakpoint 1, _start () at third.s:15
15        mov r7,#1
(gdb) x/1xh &a
0x20090:        0xfffe
(gdb) x/1xsh &a
0x20090:        u"\xfffe"
```

We observed that viewing the content of the memory using the halfword (x/1xh) is different from the signed halfword (x/1xsh), however, they both seem to provide us with the 2-byte memory with the integer -2 in hexadecimal.

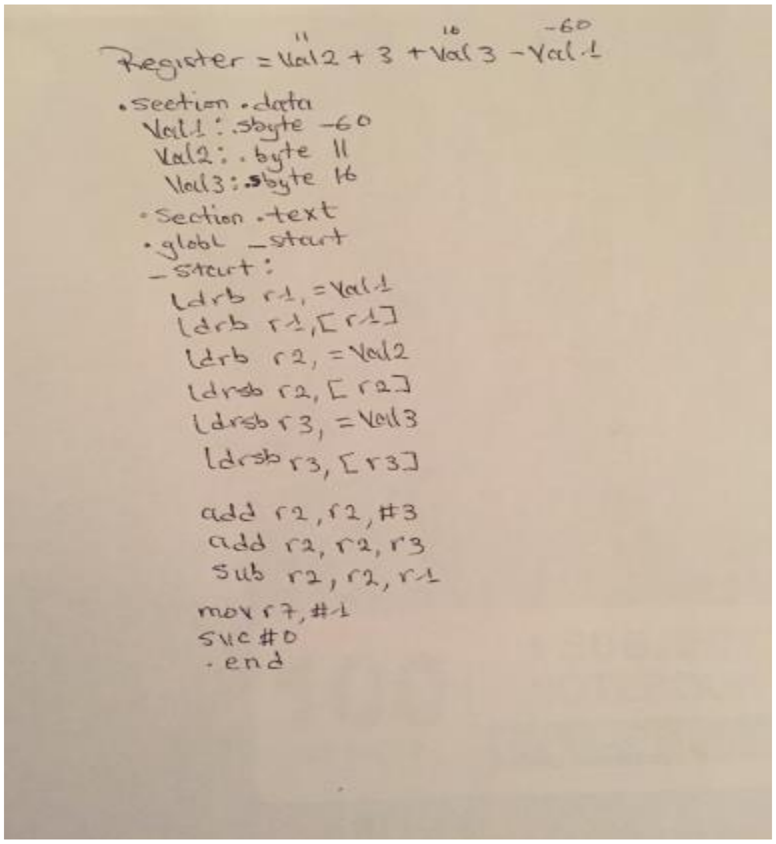
We also examine the content of the registers to make sure that everything that we loaded stayed in the registers.



```
(gdb) info registers
r0             0x1                 1
r1             0xffffffff          4294967295
r2             0xff                255
r3             0x101               257
r4             0x400               1024
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             0x7efff3c0          0x7efff3c0
lr             0x0                 0
pc             0x10088             0x10088 <_start+20>
cpsr           0x10                16
fpscr          0x0                 0
```

Part 2:

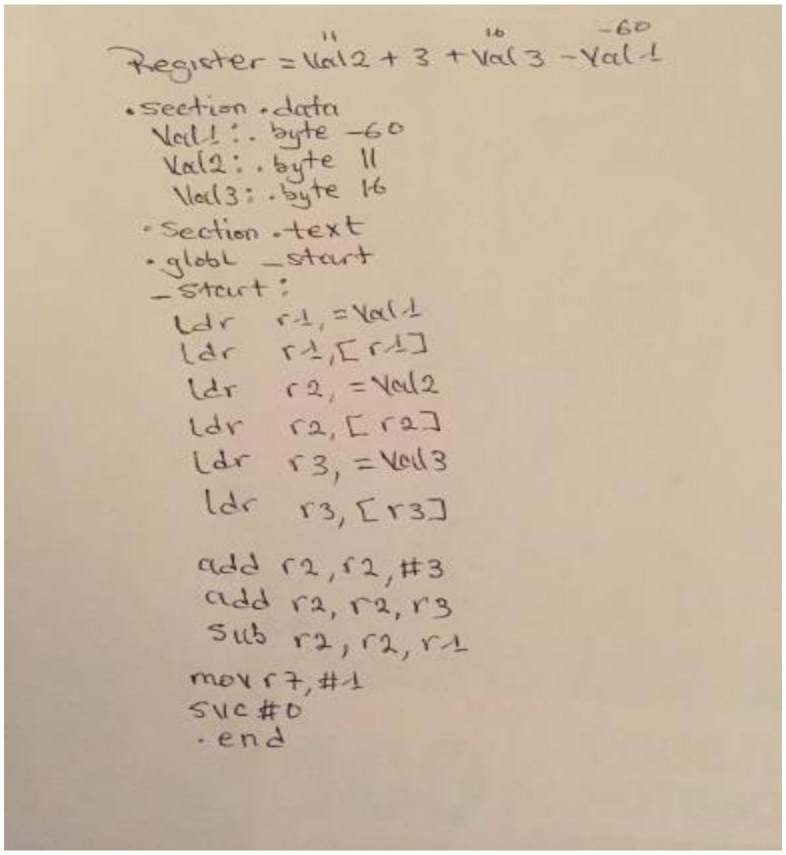
We created a second program called arithmetic3.s using Part 1 as a guide. We first planned what our program should do using the equation given as **Register = val2 + 3 + val3 - val1.** Given that Val1 is initialized with with -60, we automatically assume that we are dealing with a signed integer, so on a piece of paper, we illustrated what the program should look like.



$$\text{Register} = \overset{11}{\text{Val2}} + 3 + \overset{16}{\text{Val3}} - \overset{-60}{\text{Val.1}}$$

```
.section .data
  Val1 : .sbyte -60
   Val2 : . byte 11
    Val3 : .sbyte 16
 .Section .text
 .globl _start
  _start :
   ldrb r1, =Val1
   ldrb r1,[r1]
    ldrb  r2, =Val2
    ldrsb r2, [r2]
    ldrsb r3, =Val3
     ldrsb r3, [r3]

     add r2,r2,#3
      add r2, r2, r3
      sub  r2, r2, r1
     mov r7, #1
     svc #0
     . end
```

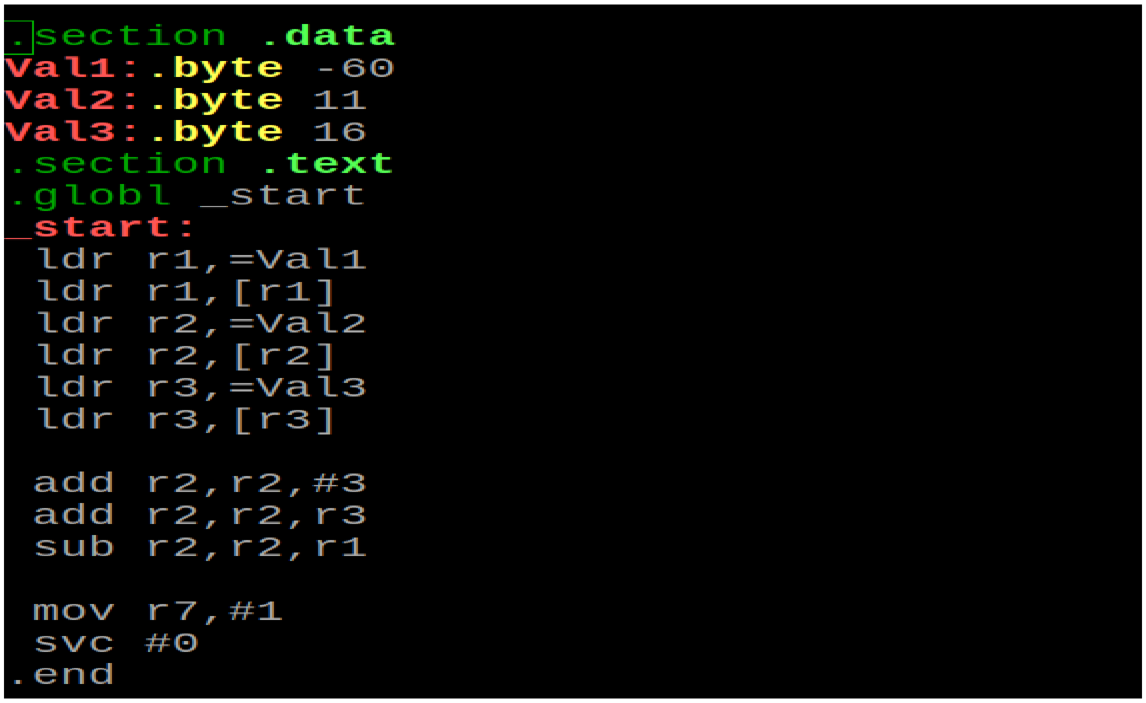
However, we got an error while trying to load the registers as a signed integer using **ldrsb** as suggested in the handout. As a result of trying to fix the program, we ended up with the following program:



```
             11        16     -60
Register = Val2 + 3 + Val3 - Val1

.section .data
 Val1: .byte -60
  Val2: .byte 11
   Val3: .byte 16
 .section .text
 .globl _start
 _start:
   ldr   r1, =Val1
   ldr   r1, [r1]
   ldr   r2, =Val2
   ldr   r2, [r2]
   ldr   r3, =Val3
   ldr   r3, [r3]

   add r2, r2, #3
   add r2, r2, r3
   sub r2, r2, r1
  mov r7, #1
  svc #0
  .end
```

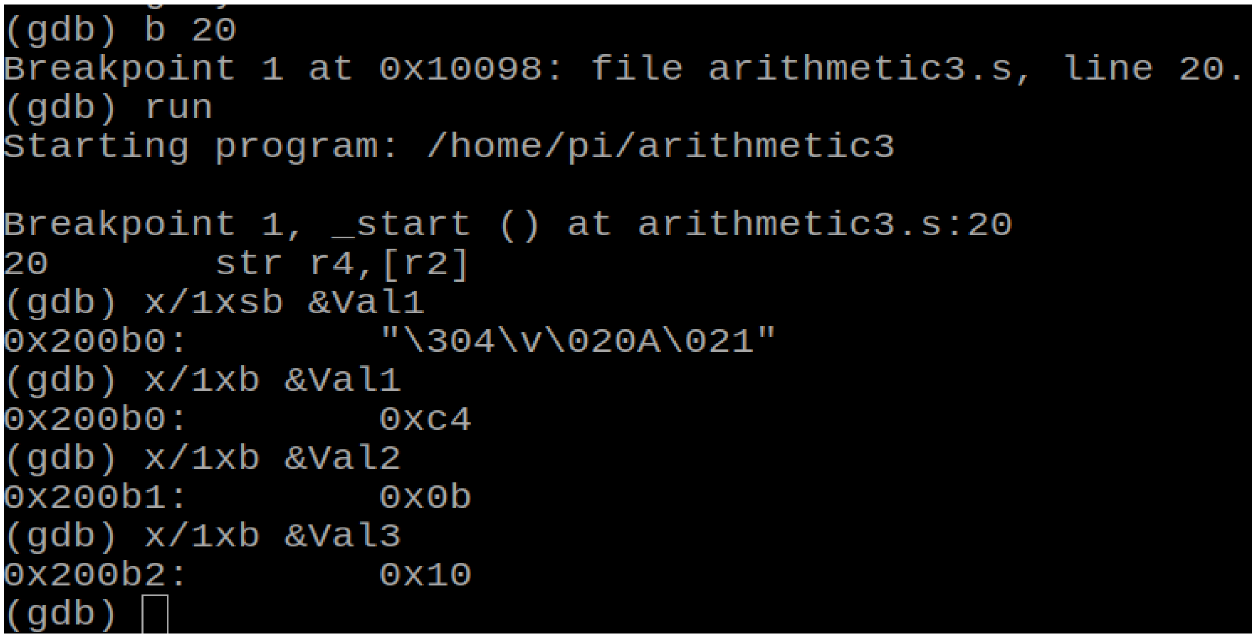
To determine the output of the program, we first opened an arithmetic3.s file to write our program. We used the directive, **.data**, to indicate that we were going to declare some variables. We created 3 byte variables: Val1, Val2, Val3. Val1 was initialized with -60, Val2 was initialized with 11, and Val3 was initialized with 16.

To use variable **val1**, we had to load (**ldr**) the memory address of **val1** into register r1. Then, we had to load the value of **val1** (**[r1]**) into register r1. After we loaded all the variables, we added **3** to the value of **r2**. We added the value of **r3** to the value of **r2**. Finally, we subtracted the value of **r1** from the value of **r2.**



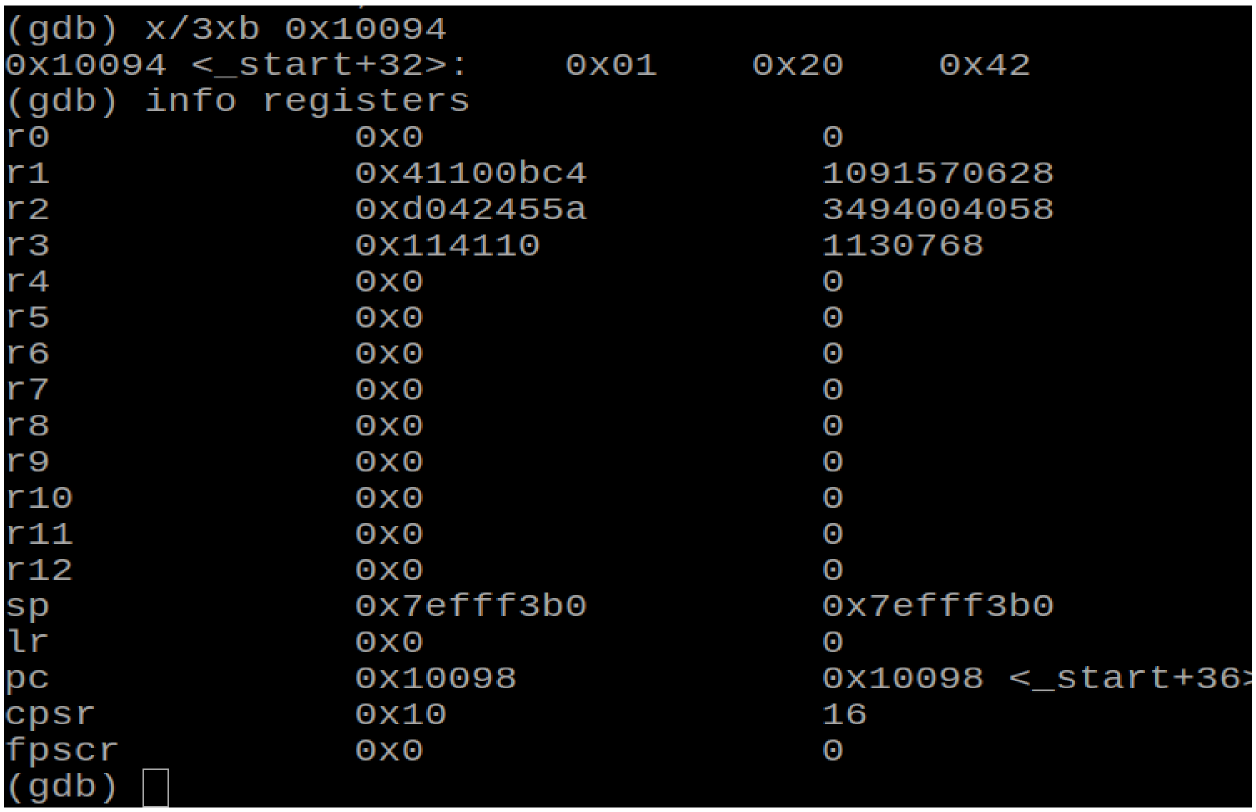
```
.section .data
Val1:.byte -60
Val2:.byte 11
Val3:.byte 16
.section .text
.globl _start
_start:
 ldr r1,=Val1
 ldr r1,[r1]
 ldr r2,=Val2
 ldr r2,[r2]
 ldr r3,=Val3
 ldr r3,[r3]

 add r2,r2,#3
 add r2,r2,r3
 sub r2,r2,r1

 mov r7,#1
 svc #0
.end
```

After the code, we assembled, linked and debugged using **(gdb arithmetic3)**. We listed the program to make sure that everything was correct, then we set a breakpoint at 20 **(b 20)** and ran the program. We examined the content of the memory using **x\1xb &Val1,Val2,Val3.**



```
(gdb) b 20
Breakpoint 1 at 0x10098: file arithmetic3.s, line 20.
(gdb) run
Starting program: /home/pi/arithmetic3

Breakpoint 1, _start () at arithmetic3.s:20
20         str r4,[r2]
(gdb) x/1xsb &Val1
0x200b0:        "\304\v\020A\021"
(gdb) x/1xb &Val1
0x200b0:        0xc4
(gdb) x/1xb &Val2
0x200b1:        0x0b
(gdb) x/1xb &Val3
0x200b2:        0x10
(gdb) 
```

We then checked the value in register 2 (hexadecimal) in **info registers** to make sure that the result from the program was correct.



```
(gdb) x/3xb 0x10094
0x10094 <_start+32>:    0x01    0x20    0x42
(gdb) info registers
r0             0x0                 0
r1             0x41100bc4          1091570628
r2             0xd042455a          3494004058
r3             0x114110            1130768
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             0x7efff3b0          0x7efff3b0
lr             0x0                 0
pc             0x10098             0x10098 <_start+36
cpsr           0x10                16
fpscr          0x0                 0
(gdb) 
```