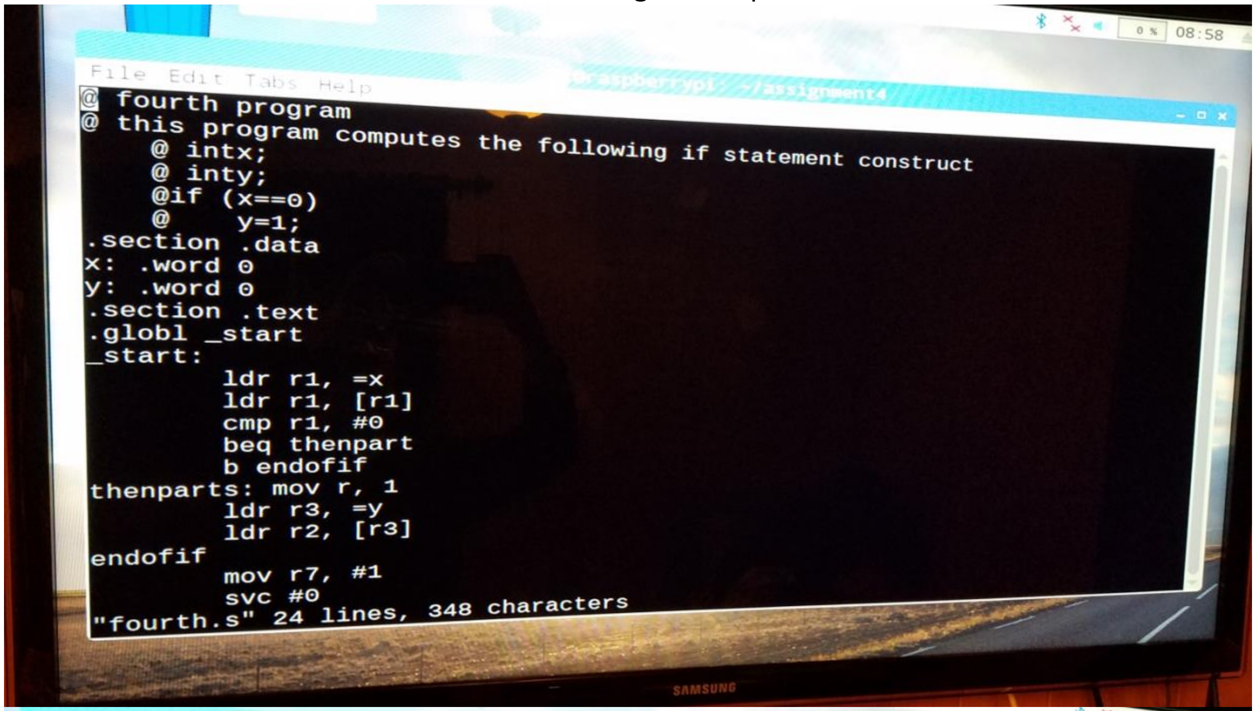
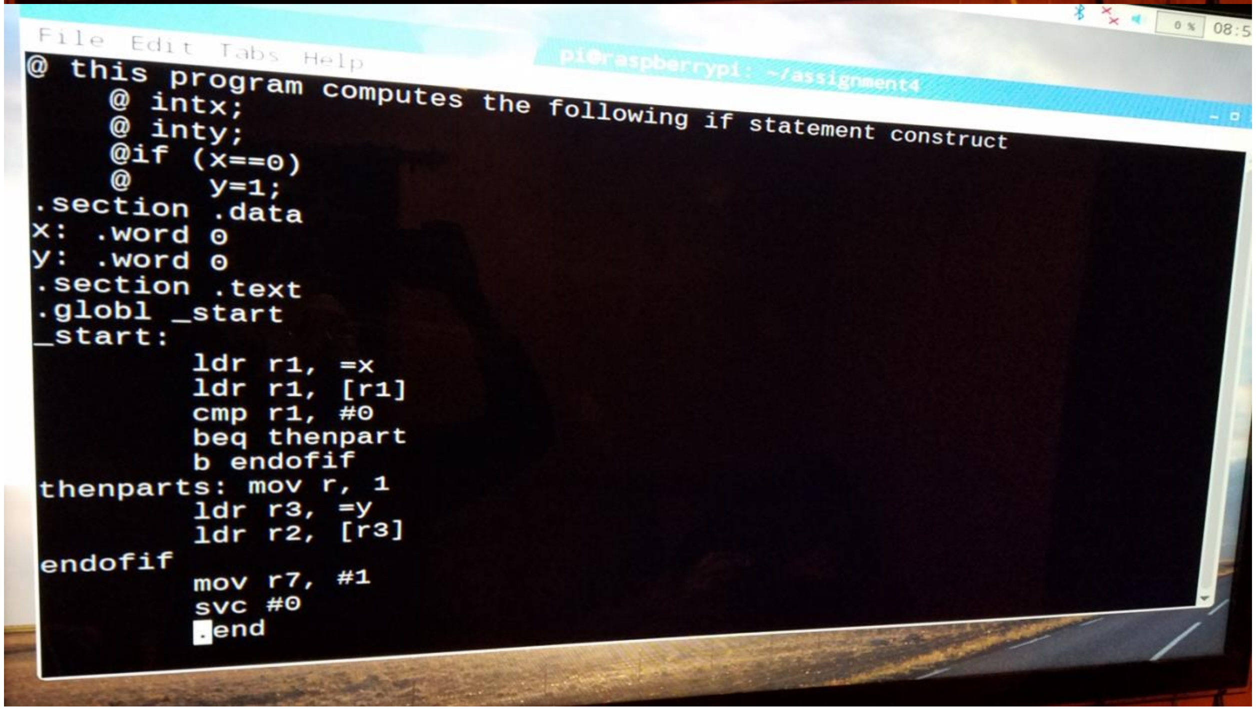
For the ARM programming assignment, I first went through the provided tutorial. I copied the code from the tutorial and came across a few bugs in the process.

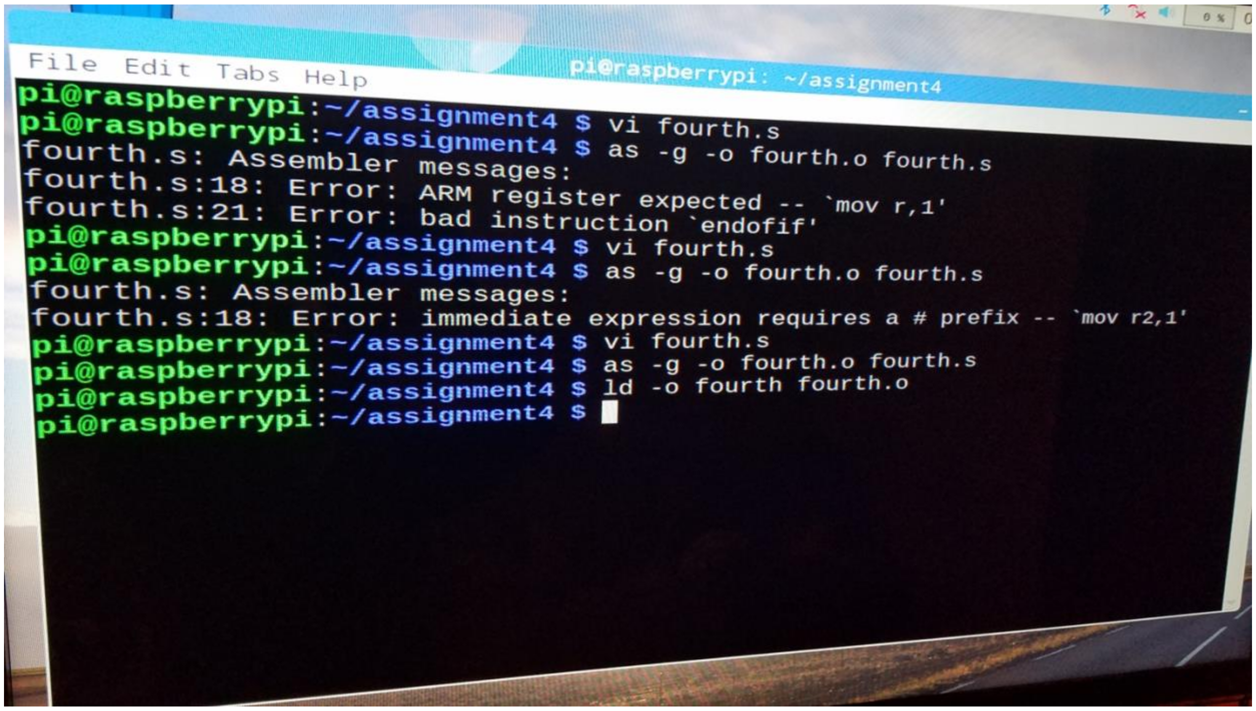


```
@ fourth program
@ this program computes the following if statement construct
   @ intx;
   @ inty;
   @if (x==0)
   @     y=1;
.section .data
x: .word 0
y: .word 0
.section .text
.globl _start
_start:
        ldr r1, =x
        ldr r1, [r1]
        cmp r1, #0
        beq thenpart
        b endofif
thenparts: mov r, 1
        ldr r3, =y
        ldr r2, [r3]
endofif
        mov r7, #1
        svc #0
"fourth.s" 24 lines, 348 characters
```



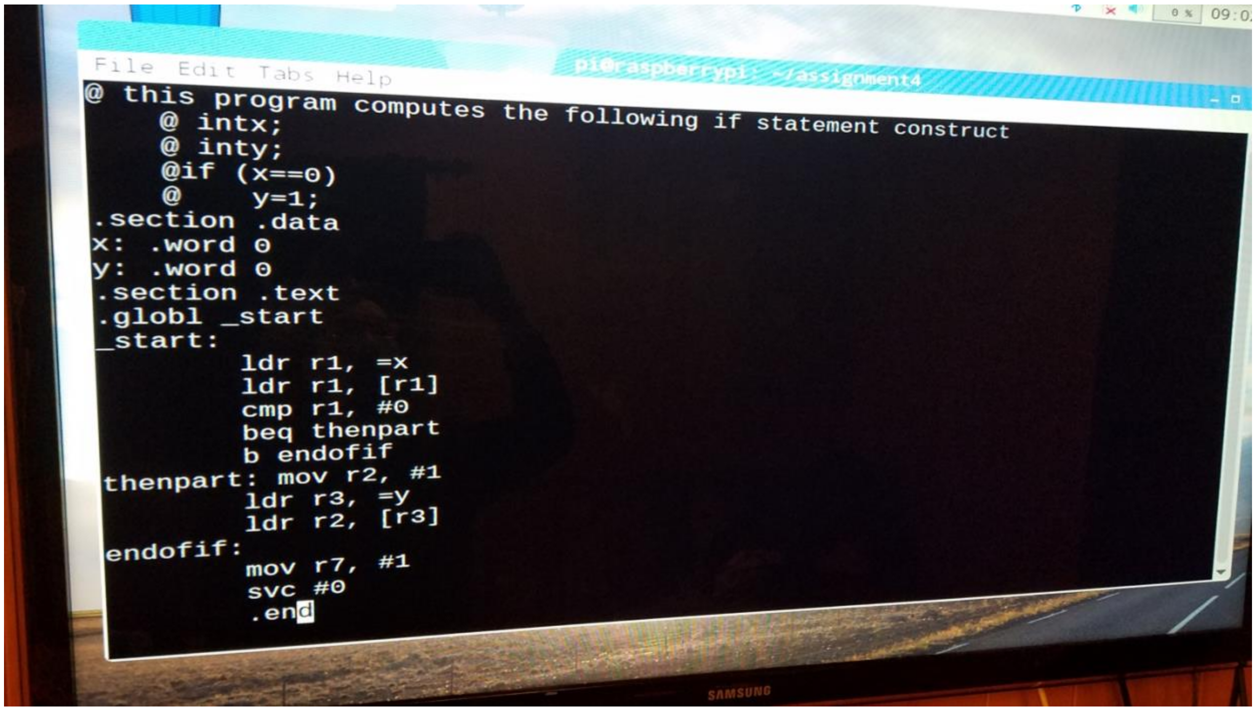
```
@ this program computes the following if statement construct
   @ intx;
   @ inty;
   @if (x==0)
   @     y=1;
.section .data
x: .word 0
y: .word 0
.section .text
.globl _start
_start:
        ldr r1, =x
        ldr r1, [r1]
        cmp r1, #0
        beq thenpart
        b endofif
thenparts: mov r, 1
        ldr r3, =y
        ldr r2, [r3]
endofif
        mov r7, #1
        svc #0
        .end
```

At first, I made a few small typos copying the code from the assignment, but I fixed those.



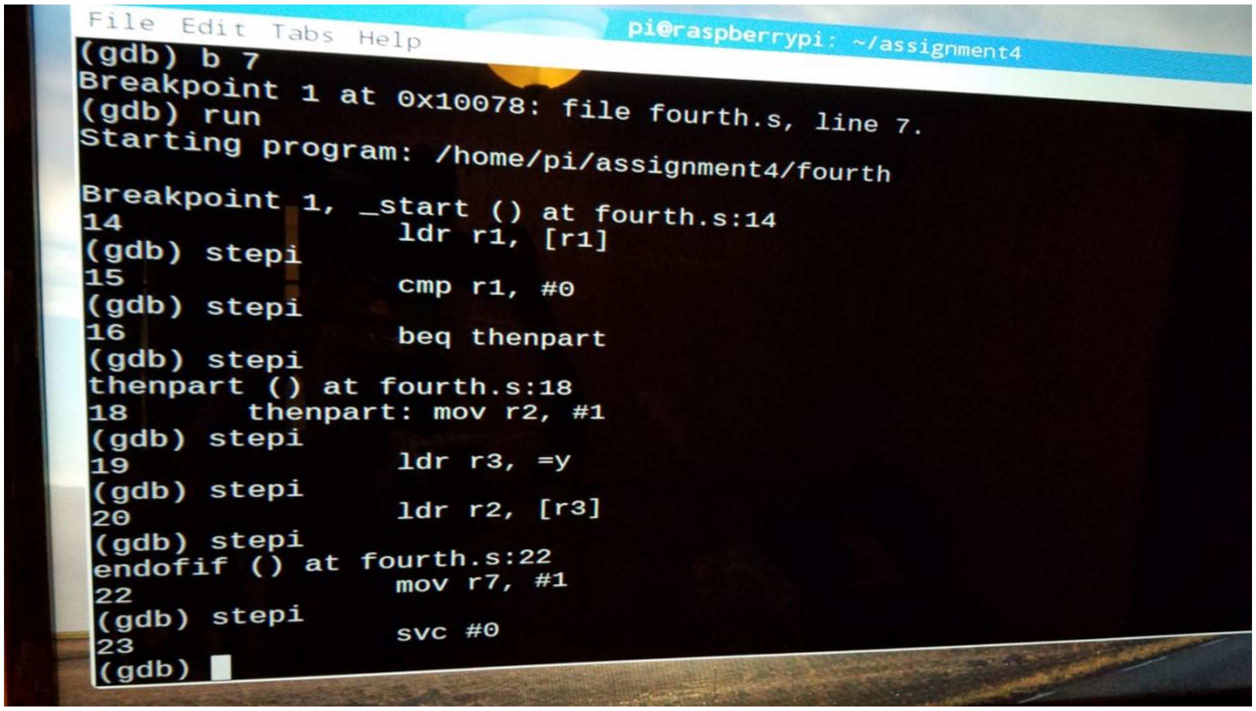
```
pi@raspberrypi:~/assignment4 $ vi fourth.s
pi@raspberrypi:~/assignment4 $ as -g -o fourth.o fourth.s
fourth.s: Assembler messages:
fourth.s:18: Error: ARM register expected -- `mov r,1'
fourth.s:21: Error: bad instruction `endofif'
pi@raspberrypi:~/assignment4 $ vi fourth.s
pi@raspberrypi:~/assignment4 $ as -g -o fourth.o fourth.s
fourth.s: Assembler messages:
fourth.s:18: Error: immediate expression requires a # prefix -- `mov r2,1'
pi@raspberrypi:~/assignment4 $ vi fourth.s
pi@raspberrypi:~/assignment4 $ as -g -o fourth.o fourth.s
pi@raspberrypi:~/assignment4 $ ld -o fourth fourth.o
pi@raspberrypi:~/assignment4 $ 
```

After fixing my typos, the assembler still threw an error. Luckily the assembler also told us how to correct the syntax error and, in this case, the syntactic correction also gave us the semantic solution we were looking for.



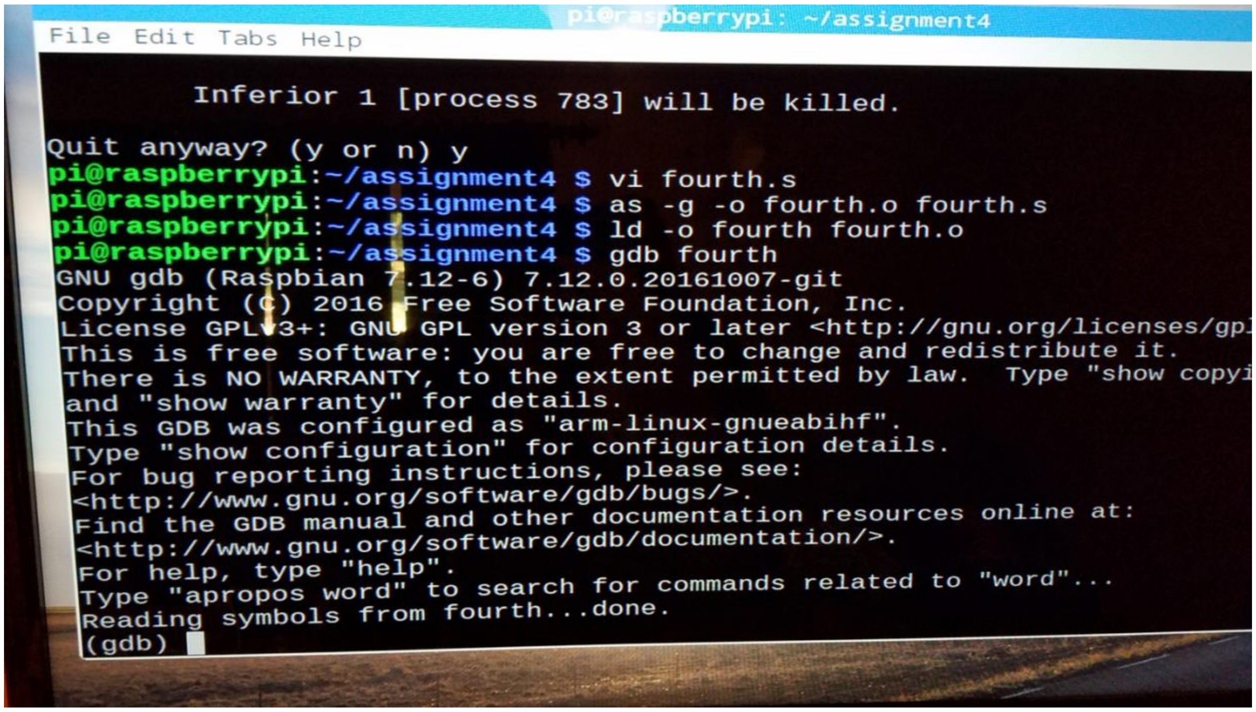
```
@ this program computes the following if statement construct
    @ intx;
    @ inty;
    @if (x==0)
    @     y=1;
.section .data
x: .word 0
y: .word 0
.section .text
.globl _start
_start:
         ldr r1, =x
         ldr r1, [r1]
         cmp r1, #0
         beq thenpart
         b endofif
thenpart: mov r2, #1
         ldr r3, =y
         ldr r2, [r3]
endofif:
         mov r7, #1
         svc #0
         .end
```

This is the the edited code (the sample code that compiled correctly).



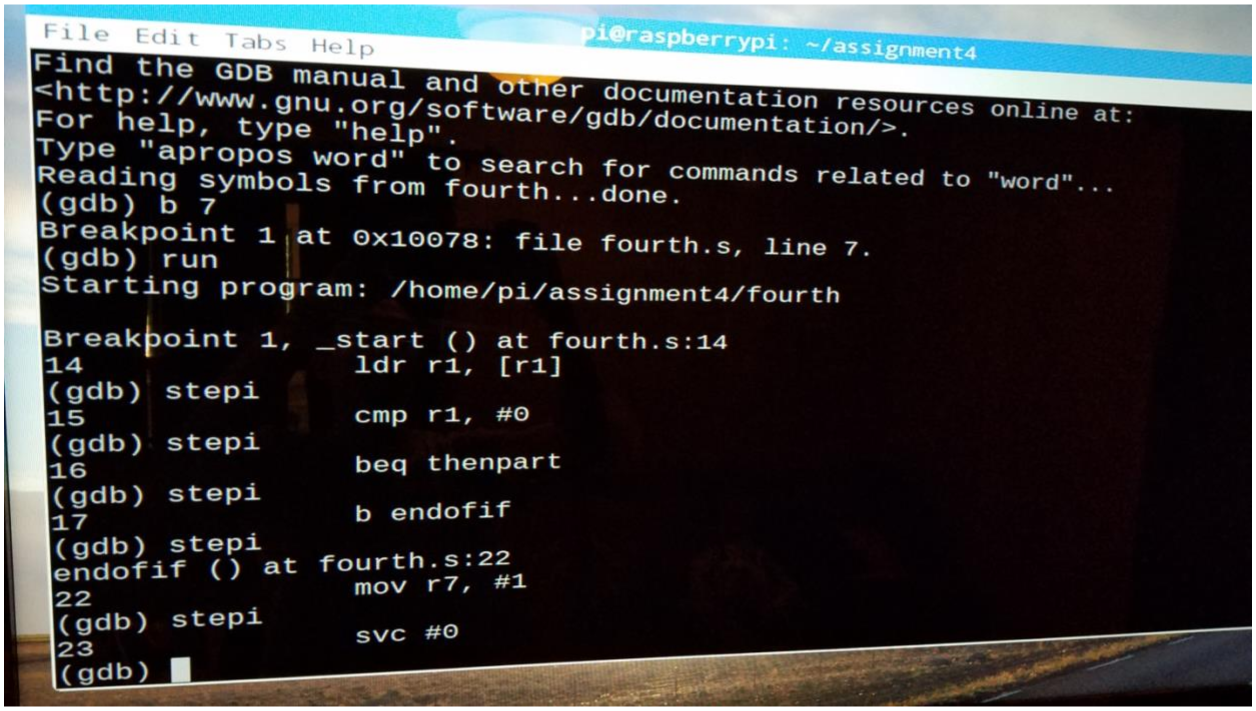
```
(gdb) b 7
Breakpoint 1 at 0x10078: file fourth.s, line 7.
(gdb) run
Starting program: /home/pi/assignment4/fourth

Breakpoint 1, _start () at fourth.s:14
14              ldr r1, [r1]
(gdb) stepi
15              cmp r1, #0
(gdb) stepi
16              beq thenpart
(gdb) stepi
thenpart () at fourth.s:18
18      thenpart: mov r2, #1
(gdb) stepi
19              ldr r3, =y
(gdb) stepi
20              ldr r2, [r3]
(gdb) stepi
endofif () at fourth.s:22
22              mov r7, #1
(gdb) stepi
23              svc #0
(gdb)
```

When x is set to 0, as in the example code, the code jumps from line 16 to 18 because “cmp r1,#0” is comparing 0 to 0, beq branches to “thenpart” because they are equal.



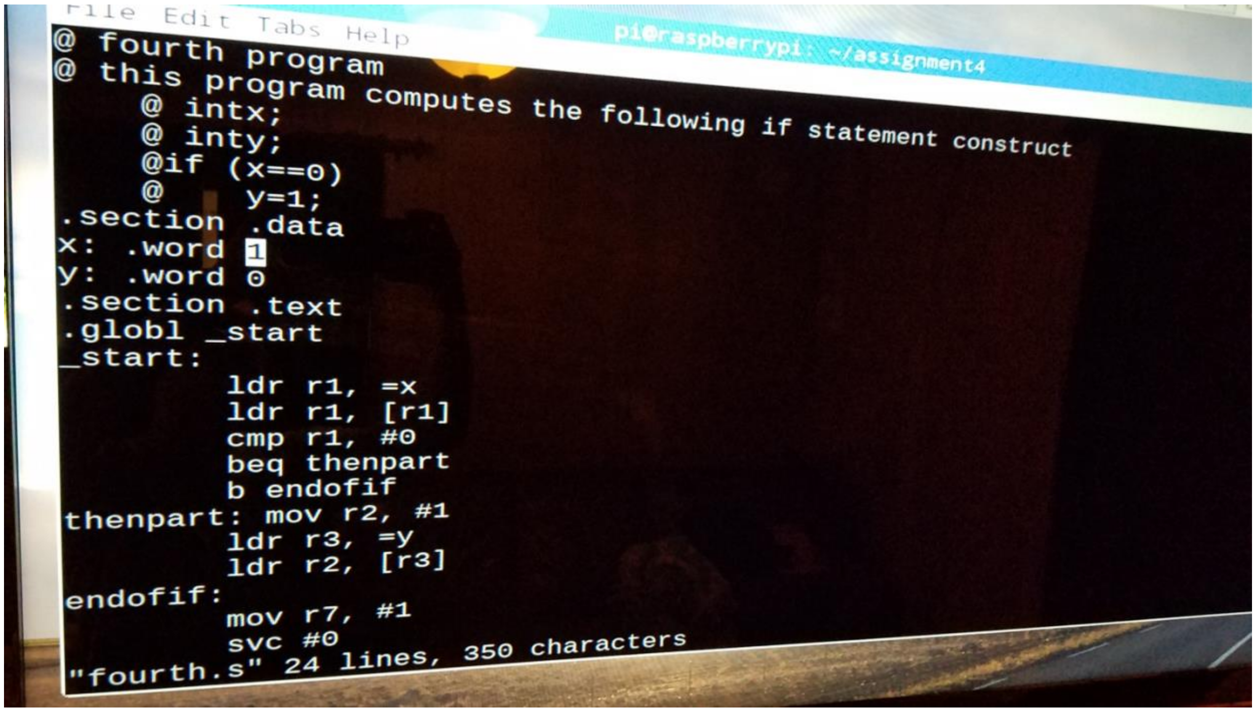
```
        Inferior 1 [process 783] will be killed.

Quit anyway? (y or n) y
pi@raspberrypi:~/assignment4 $ vi fourth.s
pi@raspberrypi:~/assignment4 $ as -g -o fourth.o fourth.s
pi@raspberrypi:~/assignment4 $ ld -o fourth fourth.o
pi@raspberrypi:~/assignment4 $ gdb fourth
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
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copyi
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 fourth...done.
(gdb)
```

In order to ensure that the branch was running properly, I tried running the code with x set to 0 as the example code shows, and I also set x to 1, to show the jump to the other branch.



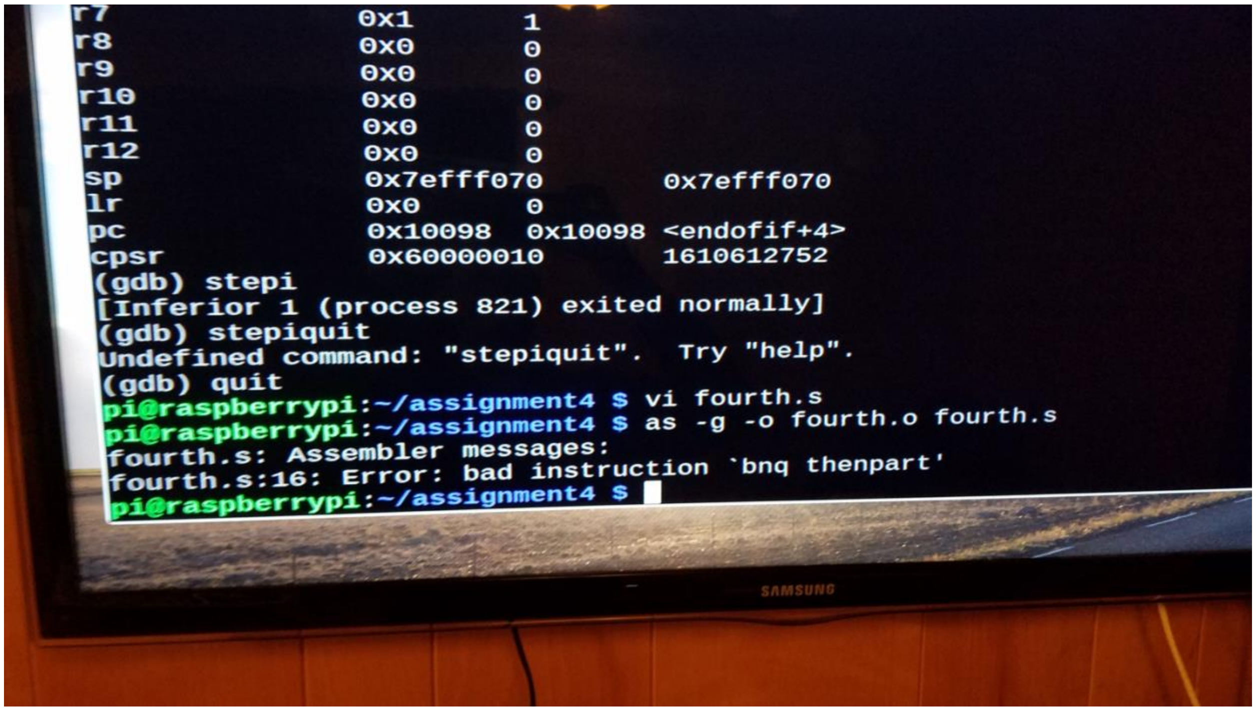
```
File  Edit  Tabs  Help                         pi@raspberrypi: ~/assignment4
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from fourth...done.
(gdb) b 7
Breakpoint 1 at 0x10078: file fourth.s, line 7.
(gdb) run
Starting program: /home/pi/assignment4/fourth

Breakpoint 1, _start () at fourth.s:14
14              ldr r1, [r1]
(gdb) stepi
15              cmp r1, #0
(gdb) stepi
16              beq thenpart
(gdb) stepi
17              b endofif
(gdb) stepi
endofif () at fourth.s:22
22              mov r7, #1
(gdb) stepi
23              svc #0
(gdb) 
```

In the code where x is set to 1, the code jumps from line 17 to 22 because “cmp r1,#0” is comparing 1 to 0, beq does not branch because they’re not equal and b branches to “endofif” because beq did not skip it. In ARM, b is an unconditional jump, so in the previous example, beq (the conditional jump) skipped it in order to run the code in “thenpart”.



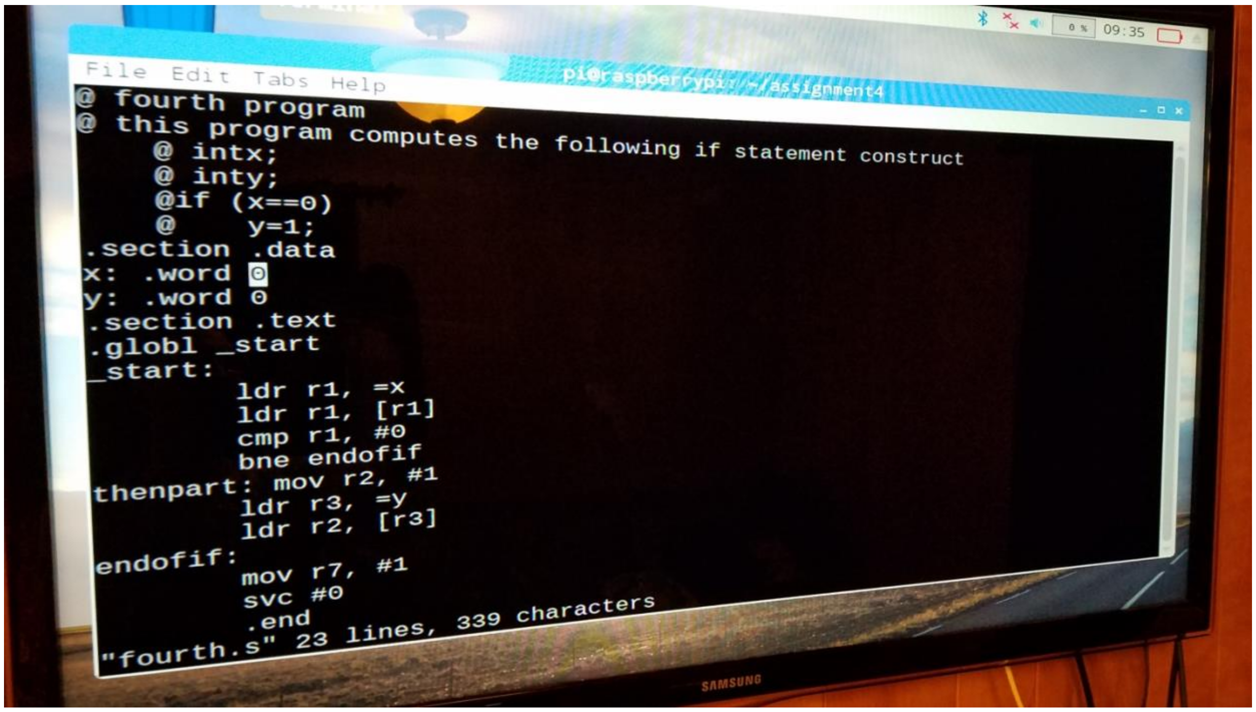
```
File  Edit  Tabs  Help                         pi@raspberrypi: ~/assignment4
@ fourth program
@ this program computes the following if statement construct
    @ intx;
    @ inty;
    @if (x==0)
    @    y=1;
.section .data
x: .word 1
y: .word 0
.section .text
.globl _start
_start:
        ldr r1, =x
        ldr r1, [r1]
        cmp r1, #0
        beq thenpart
        b endofif
thenpart: mov r2, #1
        ldr r3, =y
        ldr r2, [r3]
endofif:
        mov r7, #1
        svc #0
"fourth.s" 24 lines, 350 characters
```

This is what my test code looked like, the only change from the original code is the value of x is set to 1 instead of 0.



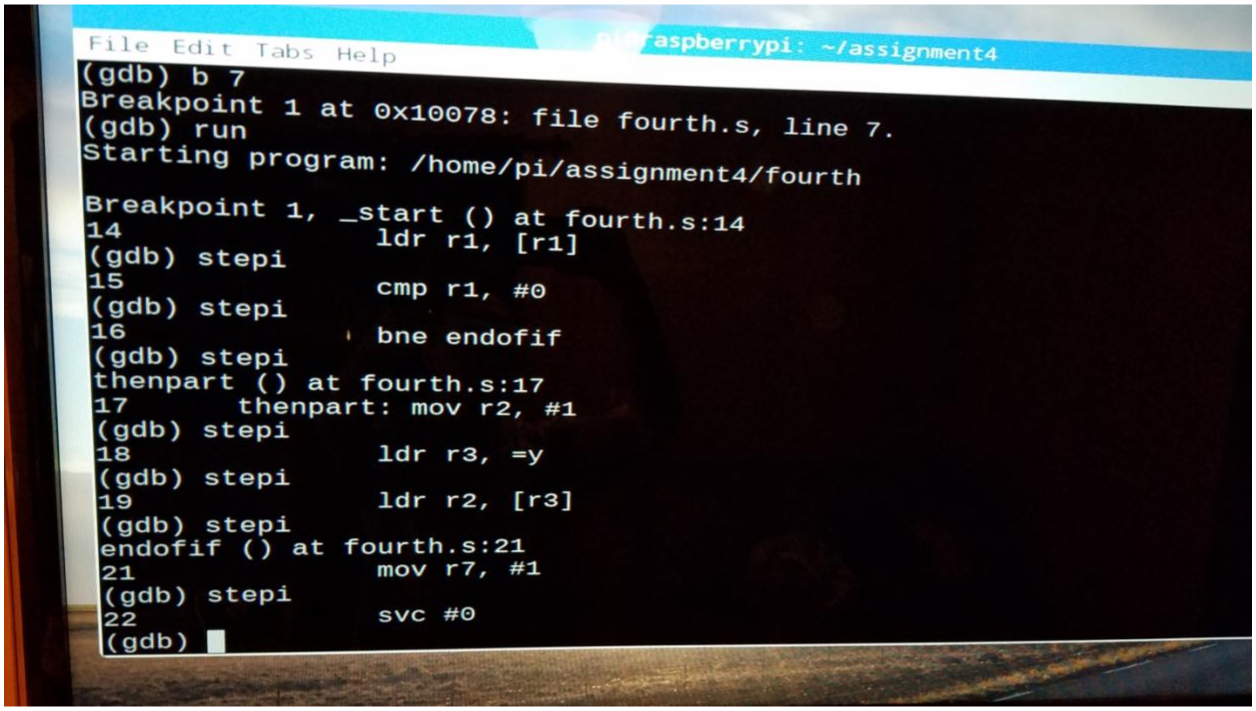
```
r7             0x1        1
r8             0x0        0
r9             0x0        0
r10            0x0        0
r11            0x0        0
r12            0x0        0
sp             0x7efff070        0x7efff070
lr             0x0        0
pc             0x10098  0x10098 <endofif+4>
cpsr           0x60000010        1610612752
(gdb) stepi
[Inferior 1 (process 821) exited normally]
(gdb) stepiquit
Undefined command: "stepiquit".  Try "help".
(gdb) quit
pi@raspberrypi:~/assignment4 $ vi fourth.s
pi@raspberrypi:~/assignment4 $ as -g -o fourth.o fourth.s
fourth.s: Assembler messages:
fourth.s:16: Error: bad instruction `bnq thenpart'
pi@raspberrypi:~/assignment4 $ 
```

After the first run through, I proceeded to update the code from part one to make it more efficient. I changed the beq to bnq and removed the b instruction, but the assembler still threw an error. After reading the appendix, I found that bnq should actually be bne.



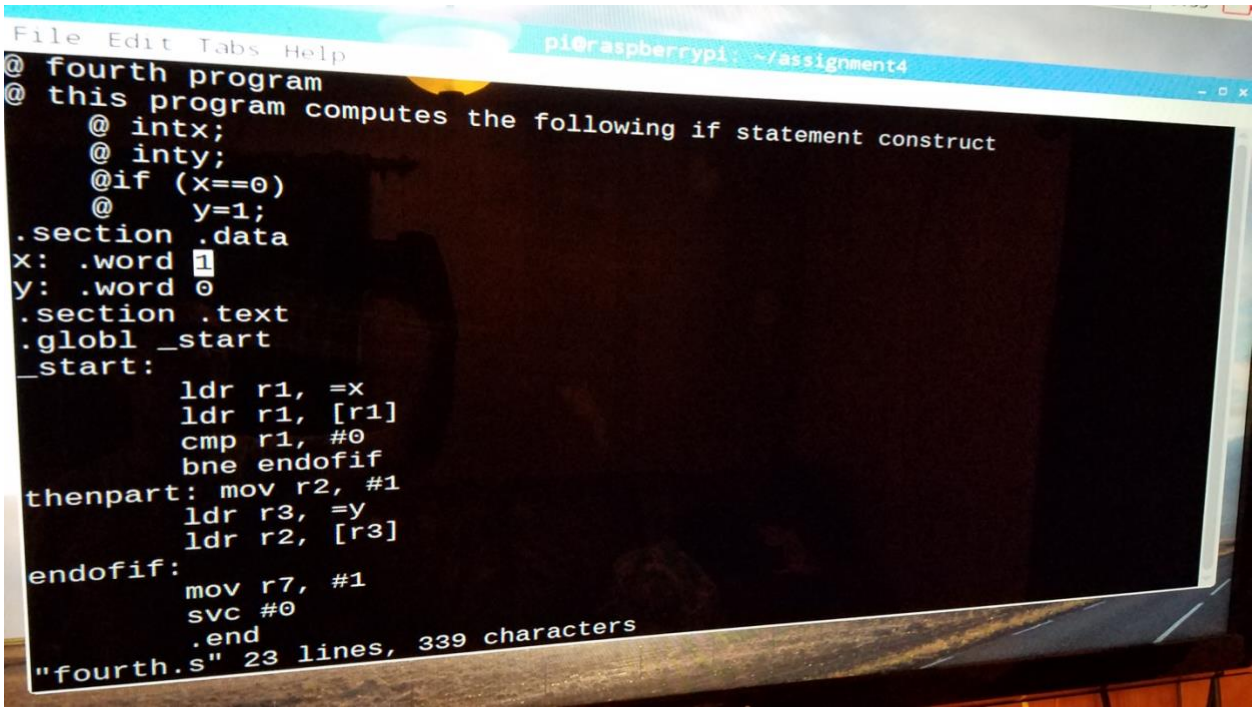
```
@ fourth program
@ this program computes the following if statement construct
      @ intx;
      @ inty;
      @if (x==0)
      @     y=1;
.section .data
x: .word 0
y: .word 0
.section .text
.globl _start
_start:
          ldr r1, =x
          ldr r1, [r1]
          cmp r1, #0
          bne endofif
thenpart: mov r2, #1
          ldr r3, =y
          ldr r2, [r3]
endofif:
          mov r7, #1
          svc #0
          .end
"fourth.s" 23 lines, 339 characters
```

In order for the code to function the same as before, the negation must be applied to the branch label as well. So this time, the branch only jumps if the comparison is not equal.



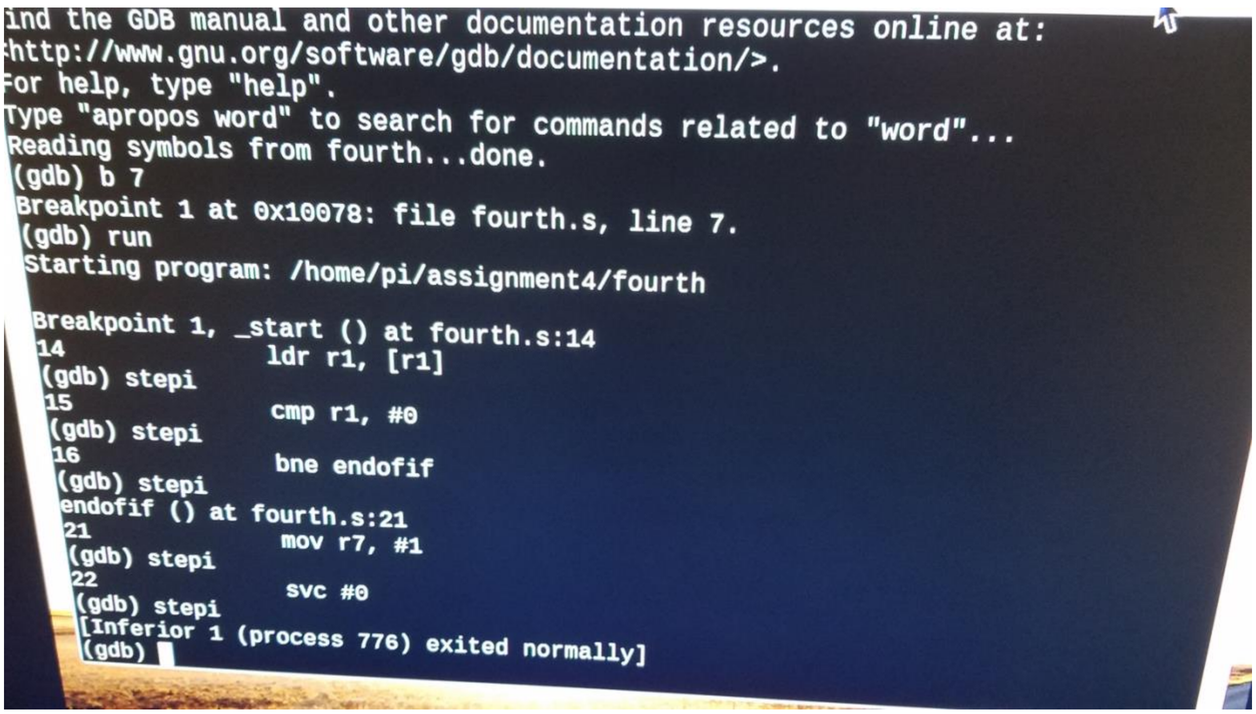
```
(gdb) b 7
Breakpoint 1 at 0x10078: file fourth.s, line 7.
(gdb) run
Starting program: /home/pi/assignment4/fourth

Breakpoint 1, _start () at fourth.s:14
14              ldr r1, [r1]
(gdb) stepi
15              cmp r1, #0
(gdb) stepi
16              bne endofif
(gdb) stepi
thenpart () at fourth.s:17
17      thenpart: mov r2, #1
(gdb) stepi
18              ldr r3, =y
(gdb) stepi
19              ldr r2, [r3]
(gdb) stepi
endofif () at fourth.s:21
21              mov r7, #1
(gdb) stepi
22              svc #0
(gdb)
```

By changing the beq to bne and removing the b after it, the code now works the same way as before but more efficiently. The cmp is still comparing 0 to 0, but because the bne only branches if the numbers are not equal, the code continues running through the “thenpart” label and assigns 1 to y.



```
@ fourth program
@ this program computes the following if statement construct
    @ intx;
    @ inty;
    @if (x==0)
    @     y=1;
.section .data
x: .word 1
y: .word 0
.section .text
.globl _start
_start:
        ldr r1, =x
        ldr r1, [r1]
        cmp r1, #0
        bne endofif
thenpart: mov r2, #1
        ldr r3, =y
        ldr r2, [r3]
endofif:
        mov r7, #1
        svc #0
        .end
"fourth.s" 23 lines, 339 characters
```

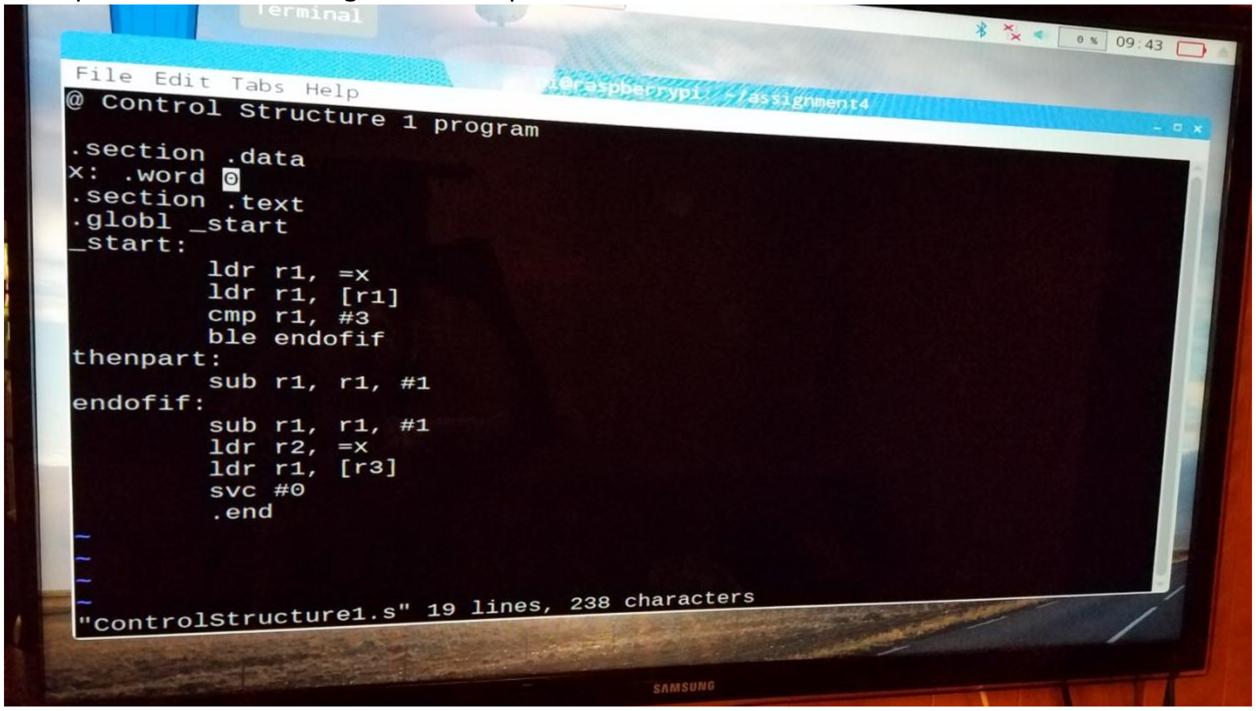
After running through the code with x set to 0, I again decided to test the branch by changing 0 to 1.



```
ind the GDB manual and other documentation resources online at:
http://www.gnu.org/software/gdb/documentation/>.
or help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from fourth...done.
(gdb) b 7
Breakpoint 1 at 0x10078: file fourth.s, line 7.
(gdb) run
Starting program: /home/pi/assignment4/fourth

Breakpoint 1, _start () at fourth.s:14
14              ldr r1, [r1]
(gdb) stepi
15              cmp r1, #0
(gdb) stepi
16              bne endofif
(gdb) stepi
endofif () at fourth.s:21
21              mov r7, #1
(gdb) stepi
22              svc #0
(gdb) stepi
[Inferior 1 (process 776) exited normally]
(gdb) 
```

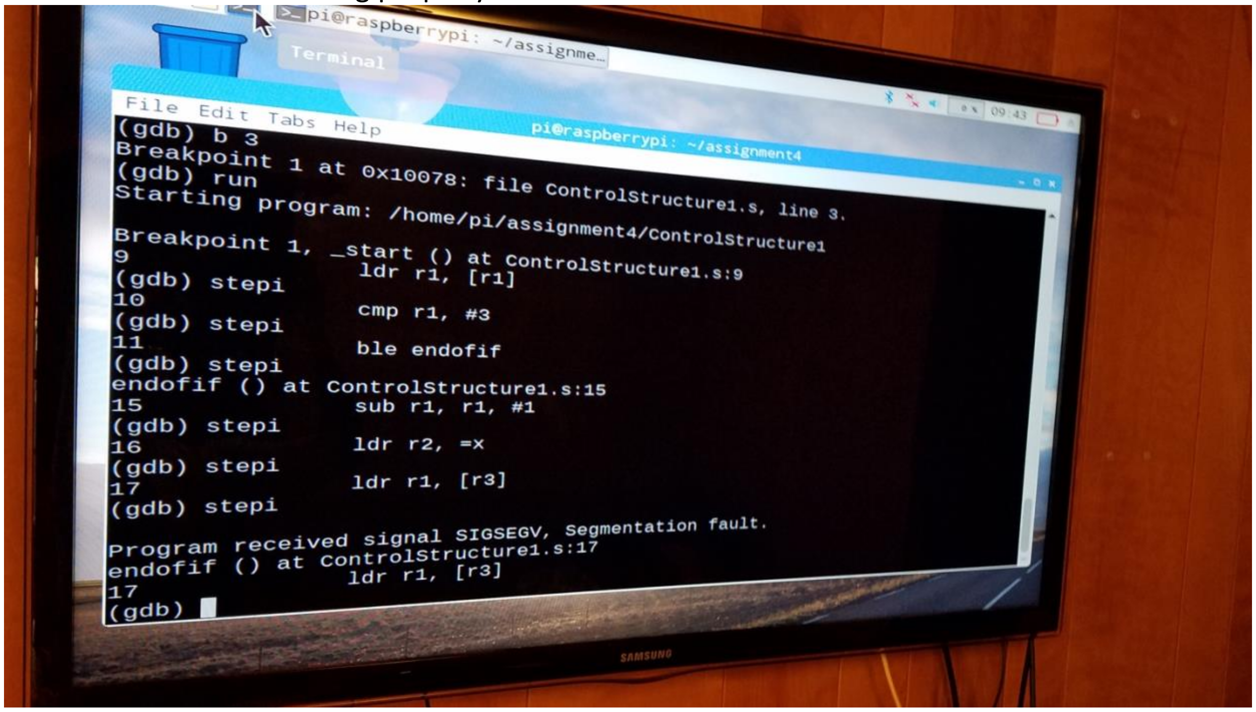
In this code, the cmp is comparing 1 to 0, and so it branches to the “endofif” and does not assign 1 to y, skipping over lines 17 to 20.

My first attempt at the control structure program, I copied the outline from the previous code example and edited the logic to the requirements.

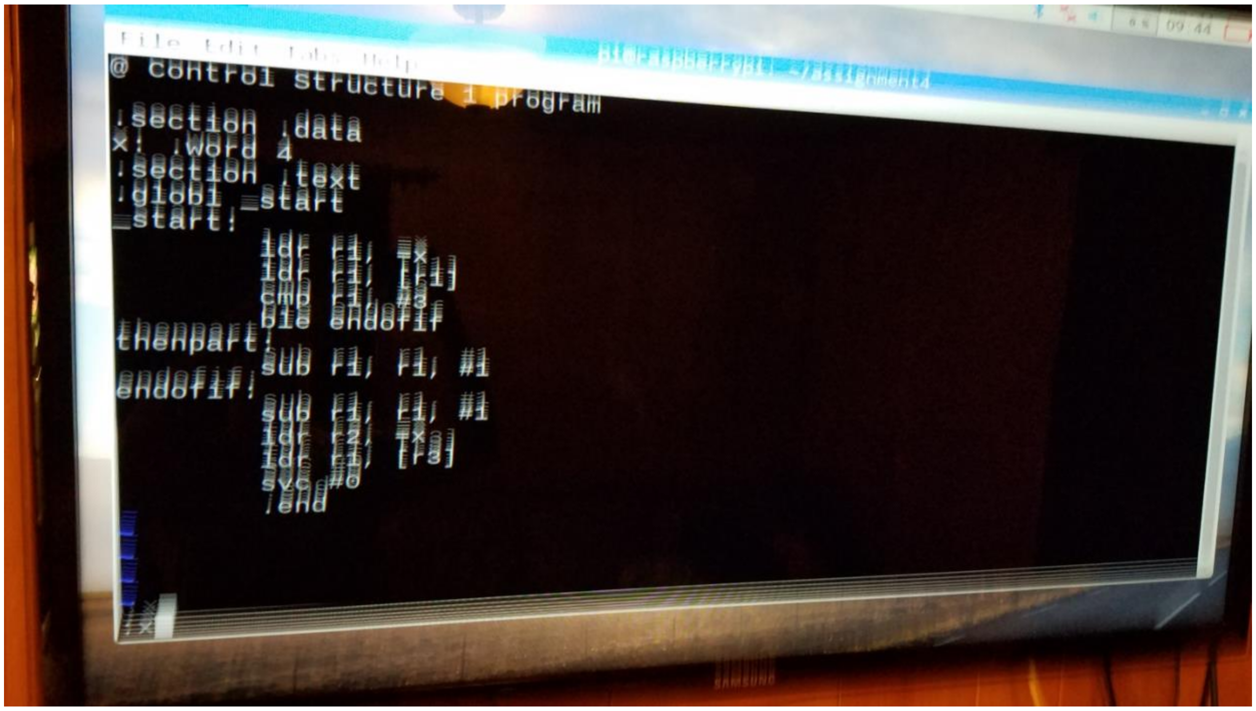


```
@ Control Structure 1 program
.section .data
x: .word 0
.section .text
.globl _start
_start:
        ldr r1, =x
        ldr r1, [r1]
        cmp r1, #3
        ble endofif
thenpart:
        sub r1, r1, #1
endofif:
        sub r1, r1, #1
        ldr r2, =x
        ldr r1, [r3]
        svc #0
        .end
~
~
~
~
"ControlStructure1.s" 19 lines, 238 characters
```

The example seems to work as expected, skipping over the “thenpart” because 0 <= 3. So the branch seems to be working properly.

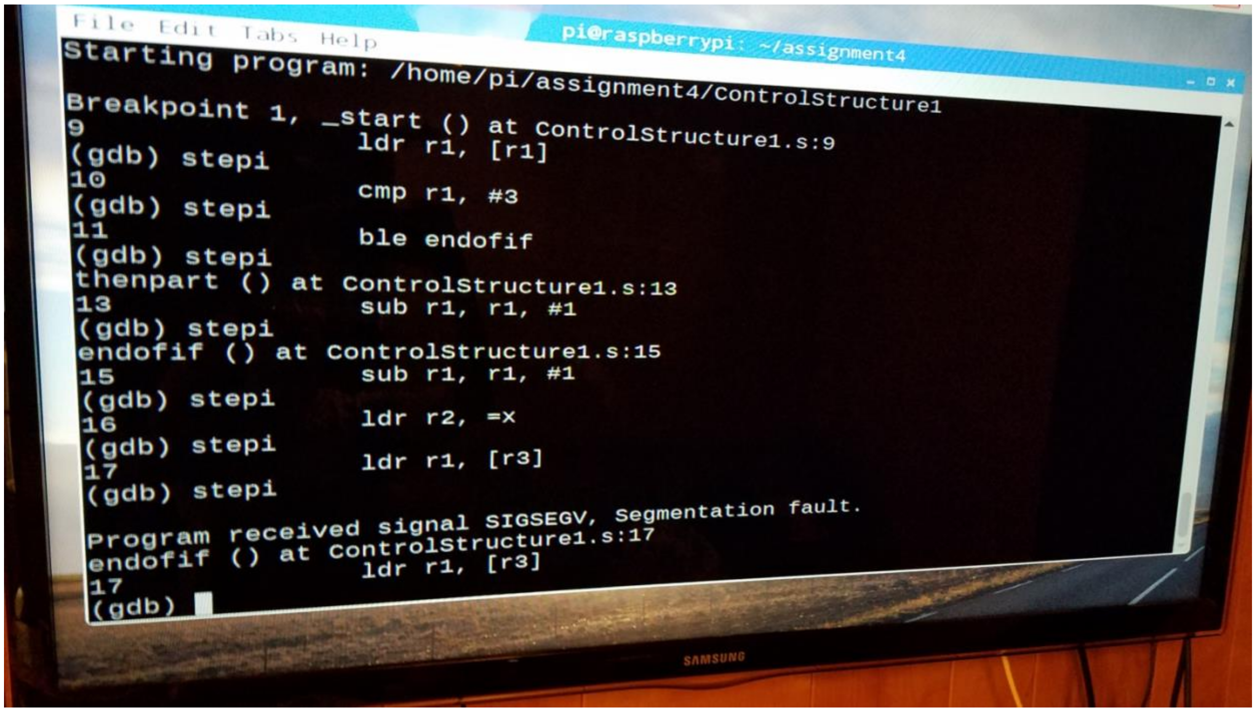


```
(gdb) b 3
Breakpoint 1 at 0x10078: file ControlStructure1.s, line 3.
(gdb) run
Starting program: /home/pi/assignment4/ControlStructure1

Breakpoint 1, _start () at ControlStructure1.s:9
9               ldr r1, [r1]
(gdb) stepi
10              cmp r1, #3
(gdb) stepi
11              ble endofif
(gdb) stepi
endofif () at ControlStructure1.s:15
15              sub r1, r1, #1
(gdb) stepi
16              ldr r2, =x
(gdb) stepi
17              ldr r1, [r3]
(gdb) stepi

Program received signal SIGSEGV, Segmentation fault.
endofif () at ControlStructure1.s:17
17              ldr r1, [r3]
(gdb) 
```



```
@ Control structure 1 program
.section .data
x: .word 4
.section .text
.globl _start
_start:
        ldr r1, =x
        ldr r1, [r1]
        cmp r1, #3
        ble endofif
thenpart:
        sub r1, r1, #1
endofif:
        sub r1, r1, #1
        ldr r2, =x
        ldr r1, [r3]
        svc #0
        .end
```

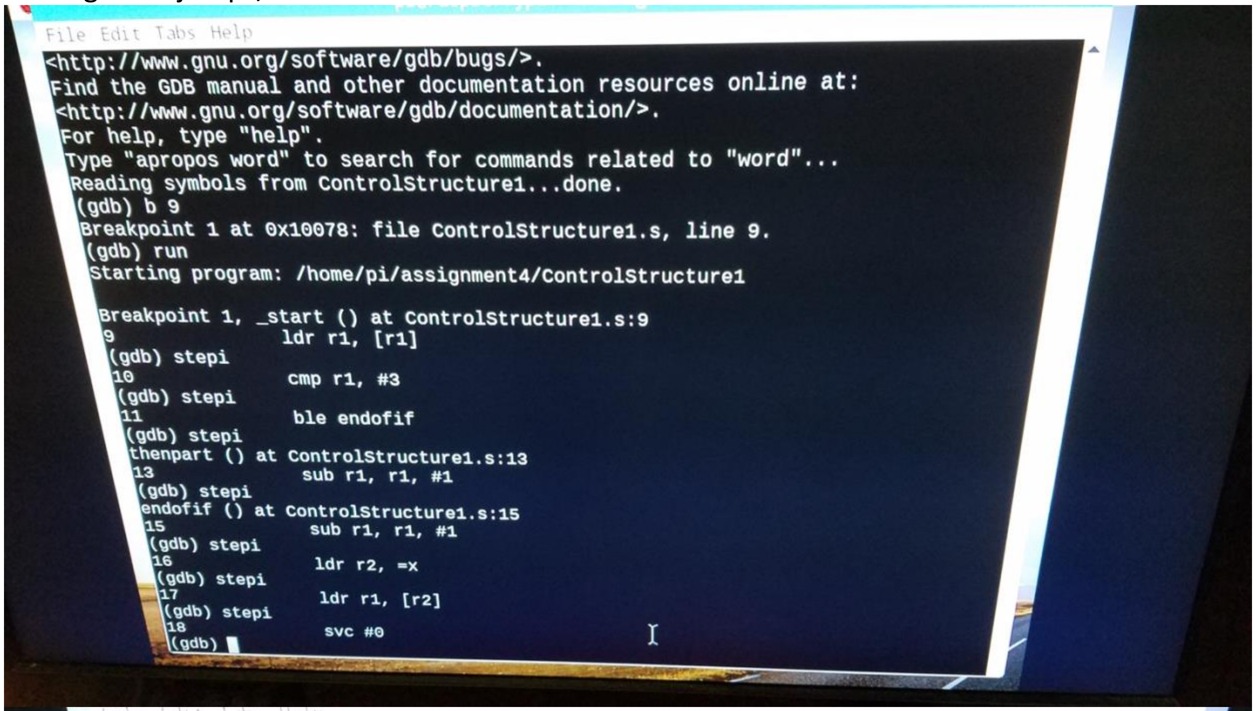
Using the same code, but changing the 0 to a 4, the code properly includes the “thenpart” label, executing lines 13 and subtracting 1 twice (the same as -2). This works as expected because 4 > 3.



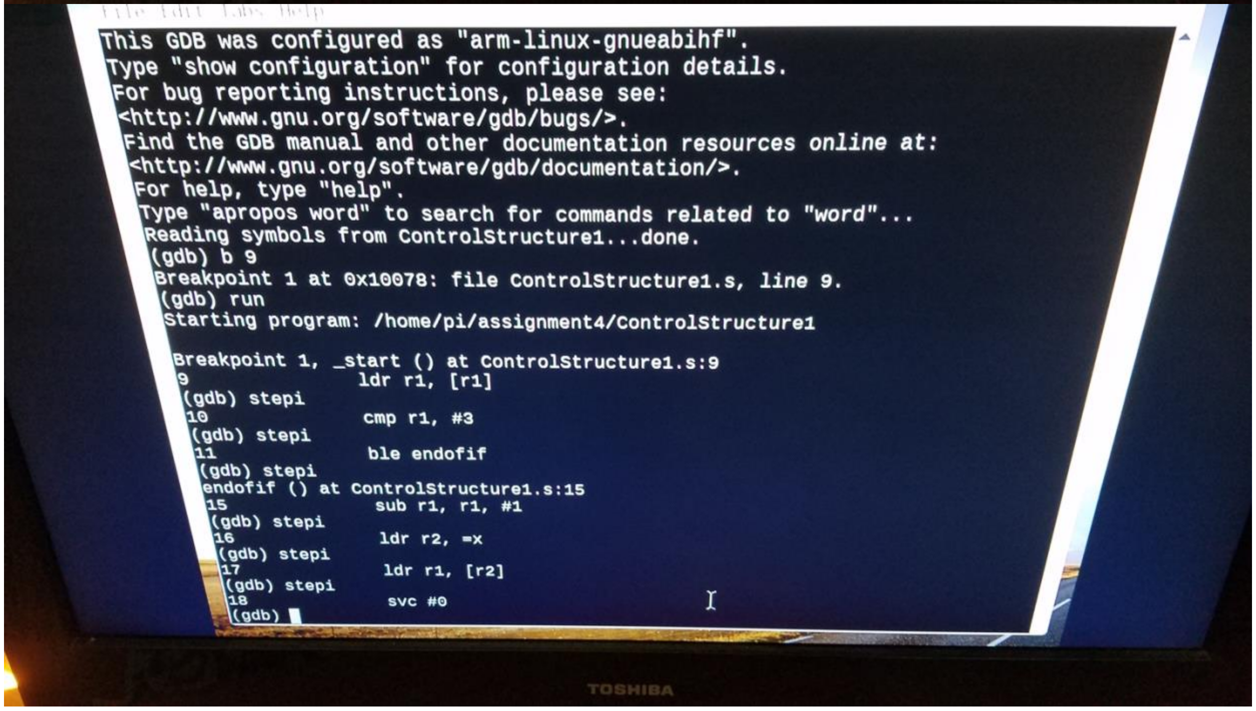
```
Starting program: /home/pi/assignment4/ControlStructure1

Breakpoint 1, _start () at ControlStructure1.s:9
9               ldr r1, [r1]
(gdb) stepi
10              cmp r1, #3
(gdb) stepi
11              ble endofif
(gdb) stepi
thenpart () at ControlStructure1.s:13
13              sub r1, r1, #1
(gdb) stepi
endofif () at ControlStructure1.s:15
15              sub r1, r1, #1
(gdb) stepi
16              ldr r2, =x
(gdb) stepi
17              ldr r1, [r3]
(gdb) stepi

Program received signal SIGSEGV, Segmentation fault.
endofif () at ControlStructure1.s:17
17              ldr r1, [r3]
(gdb) 
```

In these examples, the code did have an error at the end, loading the values into the wrong address because I had copied the r3 reference from the previous example, instead of the r2 address that I loaded my new code into. So I fixed the code from my first effort and included those screenshots below. The logic in the branch/jump did not change.

The logic the jumps/branches:

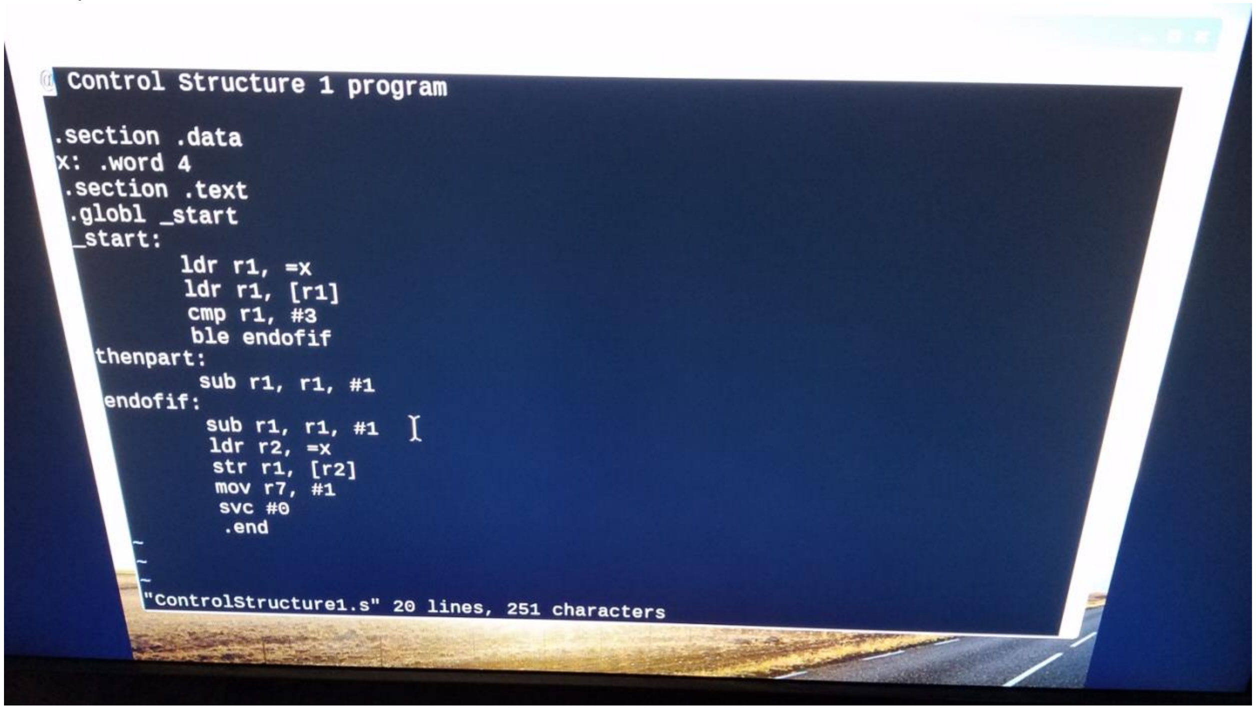


```
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ControlStructure1...done.
(gdb) b 9
Breakpoint 1 at 0x10078: file ControlStructure1.s, line 9.
(gdb) run
Starting program: /home/pi/assignment4/ControlStructure1

Breakpoint 1, _start () at ControlStructure1.s:9
9               ldr r1, [r1]
(gdb) stepi
10              cmp r1, #3
(gdb) stepi
11              ble endofif
(gdb) stepi
thenpart () at ControlStructure1.s:13
13              sub r1, r1, #1
(gdb) stepi
endofif () at ControlStructure1.s:15
15              sub r1, r1, #1
(gdb) stepi
16              ldr r2, =x
(gdb) stepi
17              ldr r1, [r2]
(gdb) stepi
18              svc #0
(gdb)
```

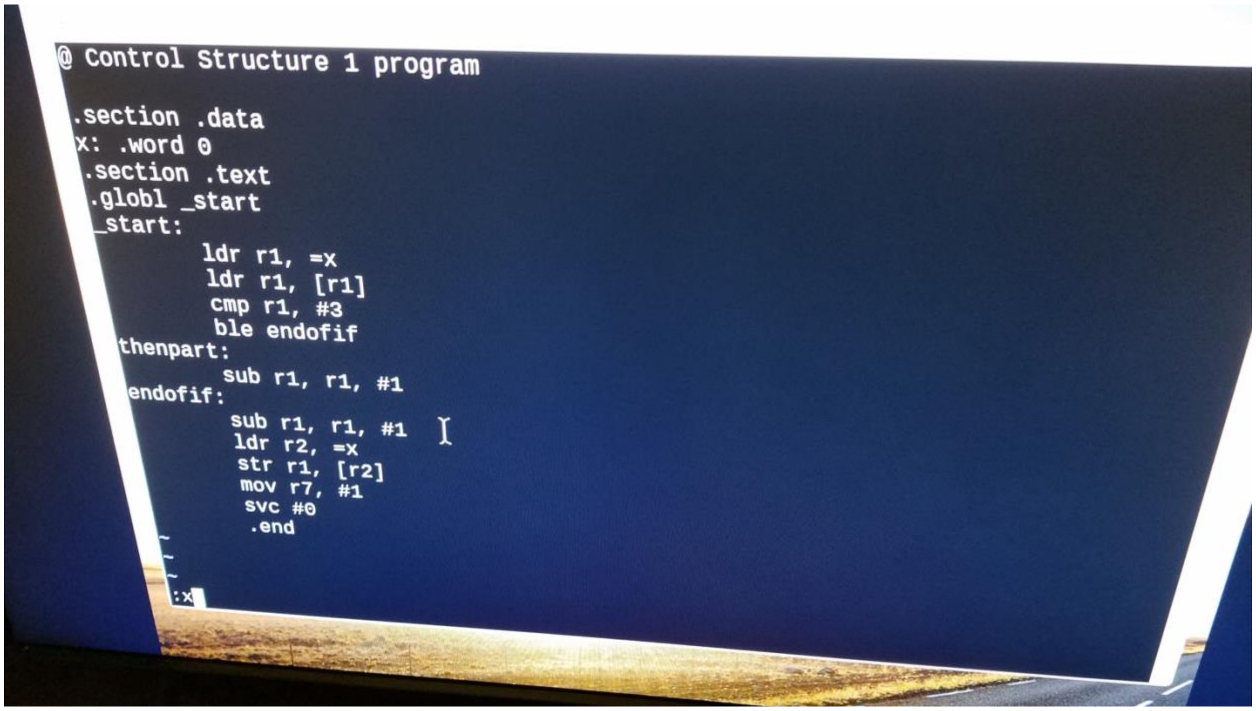


```
This GDB was configured as "arm-linux-gnueabihf".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ControlStructure1...done.
(gdb) b 9
Breakpoint 1 at 0x10078: file ControlStructure1.s, line 9.
(gdb) run
Starting program: /home/pi/assignment4/ControlStructure1

Breakpoint 1, _start () at ControlStructure1.s:9
9               ldr r1, [r1]
(gdb) stepi
10              cmp r1, #3
(gdb) stepi
11              ble endofif
(gdb) stepi
endofif () at ControlStructure1.s:15
15              sub r1, r1, #1
(gdb) stepi
16              ldr r2, =x
(gdb) stepi
17              ldr r1, [r2]
(gdb) stepi
18              svc #0
(gdb)
```

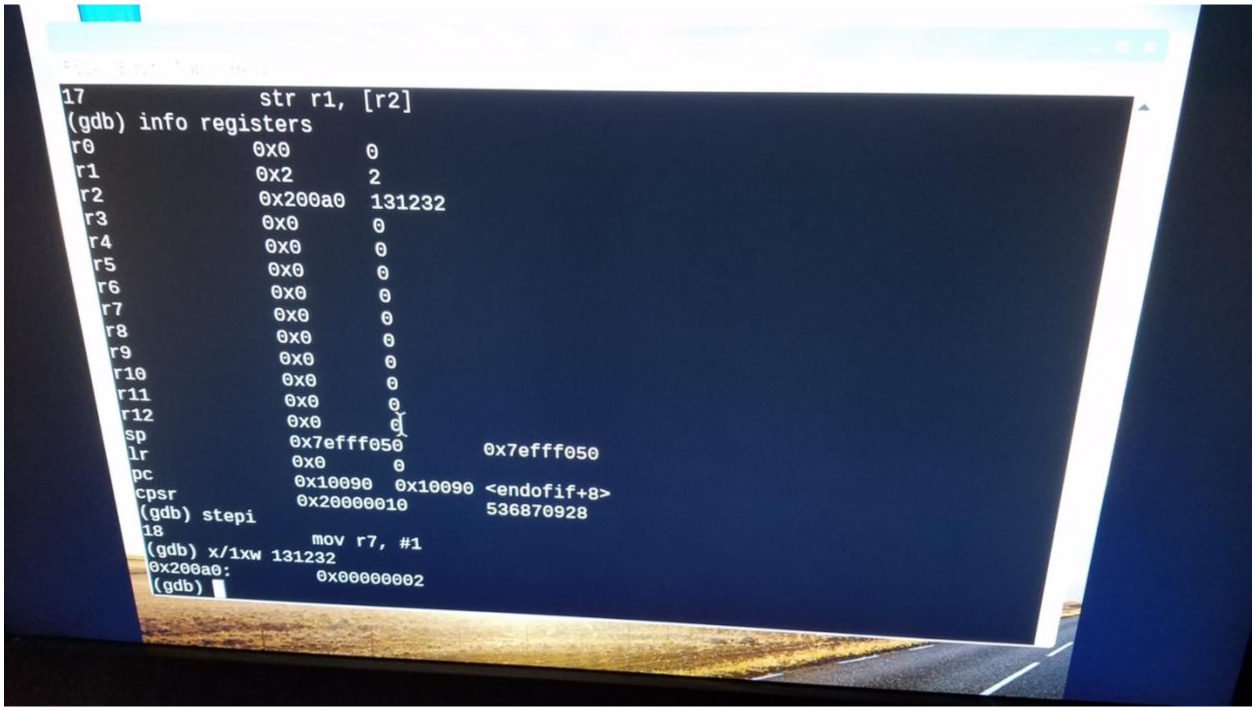
The updated code:



```
@ Control Structure 1 program

.section .data
x: .word 4
.section .text
.globl _start
_start:
        ldr r1, =x
        ldr r1, [r1]
        cmp r1, #3
        ble endofif
thenpart:
        sub r1, r1, #1
endofif:
        sub r1, r1, #1
        ldr r2, =x
        str r1, [r2]
        mov r7, #1
        svc #0
        .end
~
~
~
"ControlStructure1.s" 20 lines, 251 characters
```

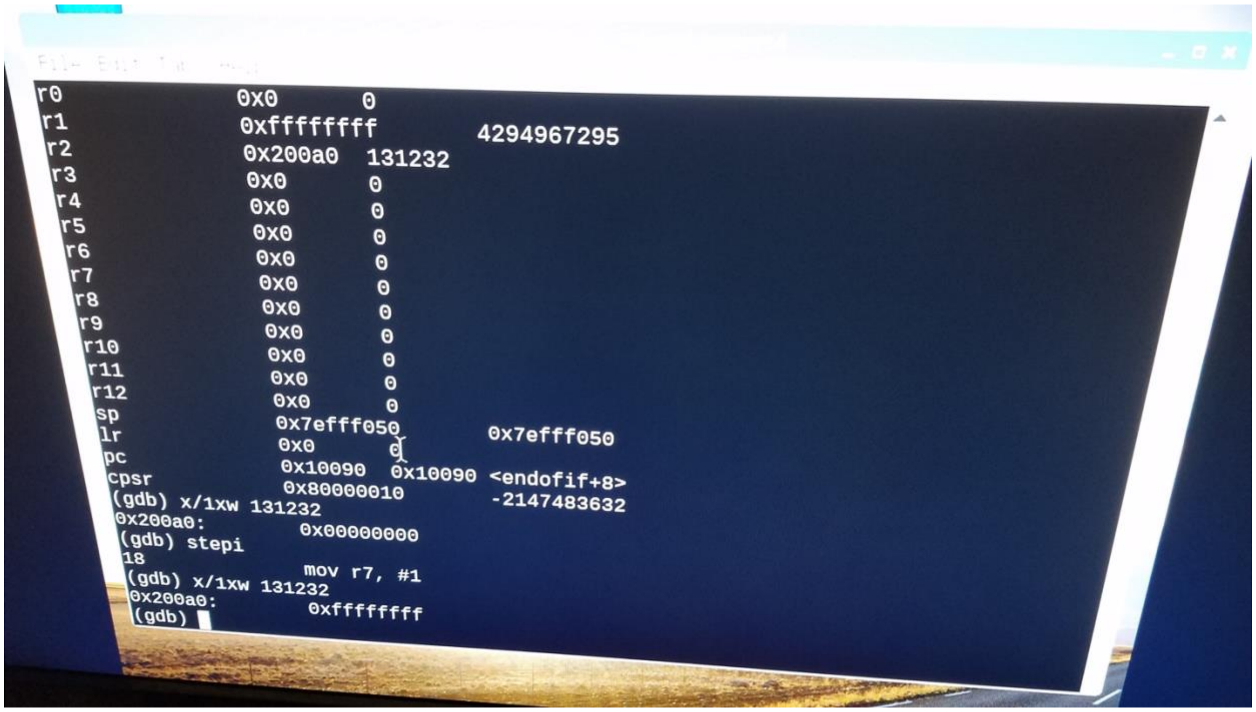


```
@ Control Structure 1 program

.section .data
x: .word 0
.section .text
.globl _start
_start:
        ldr r1, =x
        ldr r1, [r1]
        cmp r1, #3
        ble endofif
thenpart:
        sub r1, r1, #1
endofif:
        sub r1, r1, #1
        ldr r2, =x
        str r1, [r2]
        mov r7, #1
        svc #0
        .end
~
~
~
:x
```



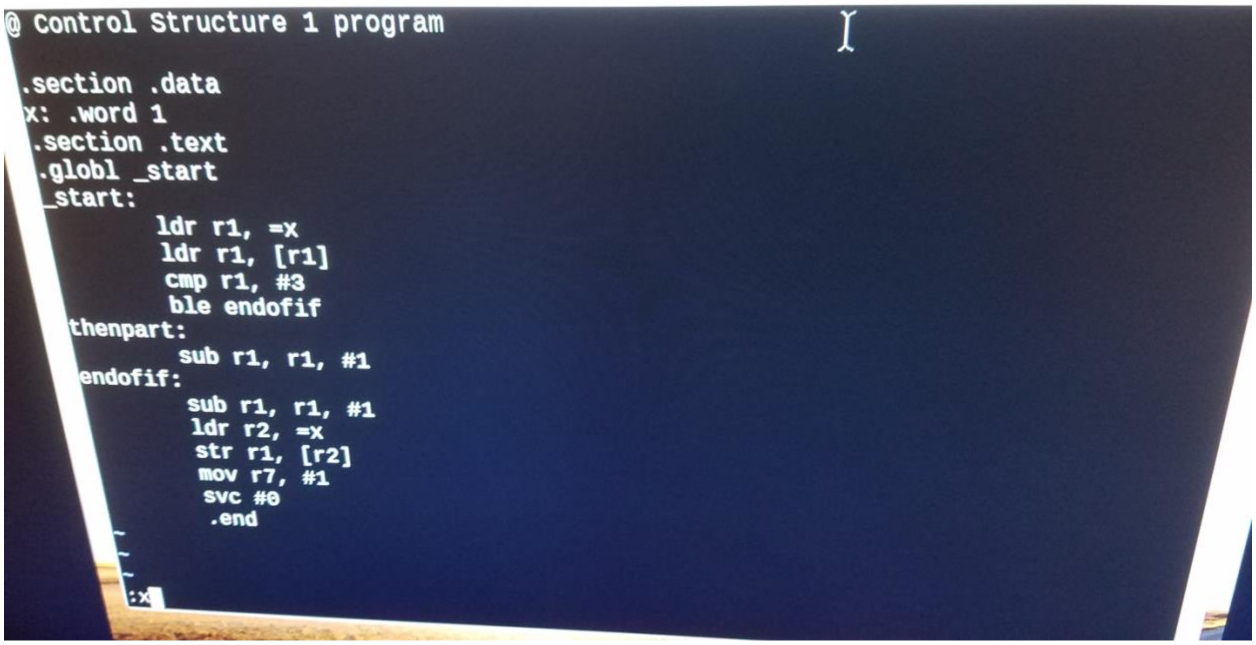
```
17              str r1, [r2]
(gdb) info registers
r0             0x0      0
r1             0x2      2
r2             0x200a0  131232
r3             0x0      0
r4             0x0      0
r5             0x0      0
r6             0x0      0
r7             0x0      0
r8             0x0      0
r9             0x0      0
r10            0x0      0
r11            0x0      0
r12            0x0      0
sp             0x7efff050       0x7efff050
lr             0x0      0
pc             0x10090  0x10090 <endofif+8>
cpsr           0x20000010       536870928
(gdb) stepi
18              mov r7, #1
(gdb) x/1xw 131232
0x200a0:        0x00000002
(gdb)
```

This example shows that the final value of 2 has been loaded to the variable x, because 4 is not less than or equal to 3 and as such, 4 minus 2 equals 2.



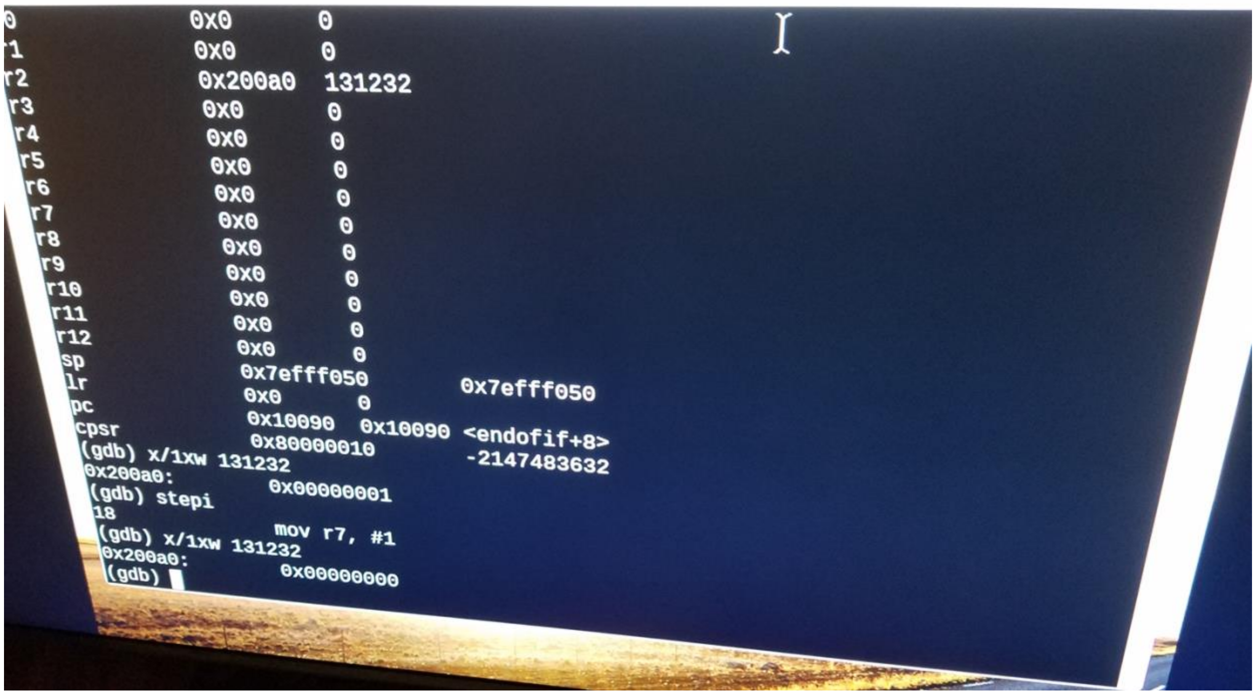
```
r0             0x0      0
r1             0xffffffff       4294967295
r2             0x200a0  131232
r3             0x0      0
r4             0x0      0
r5             0x0      0
r6             0x0      0
r7             0x0      0
r8             0x0      0
r9             0x0      0
r10            0x0      0
r11            0x0      0
r12            0x0      0
sp             0x7efff050       0x7efff050
lr             0x0      0
pc             0x10090  0x10090 <endofif+8>
cpsr           0x80000010       -2147483632
(gdb) x/1xw 131232
0x200a0:        0x00000000
(gdb) stepi
18              mov r7, #1
(gdb) x/1xw 131232
0x200a0:        0xffffffff
(gdb)
```

This example shows that the final value of -1 has been loaded to the variable x, because 0 is less than or equal to 3 and as such, 0 minus 1 equals -1 (ffffffff).

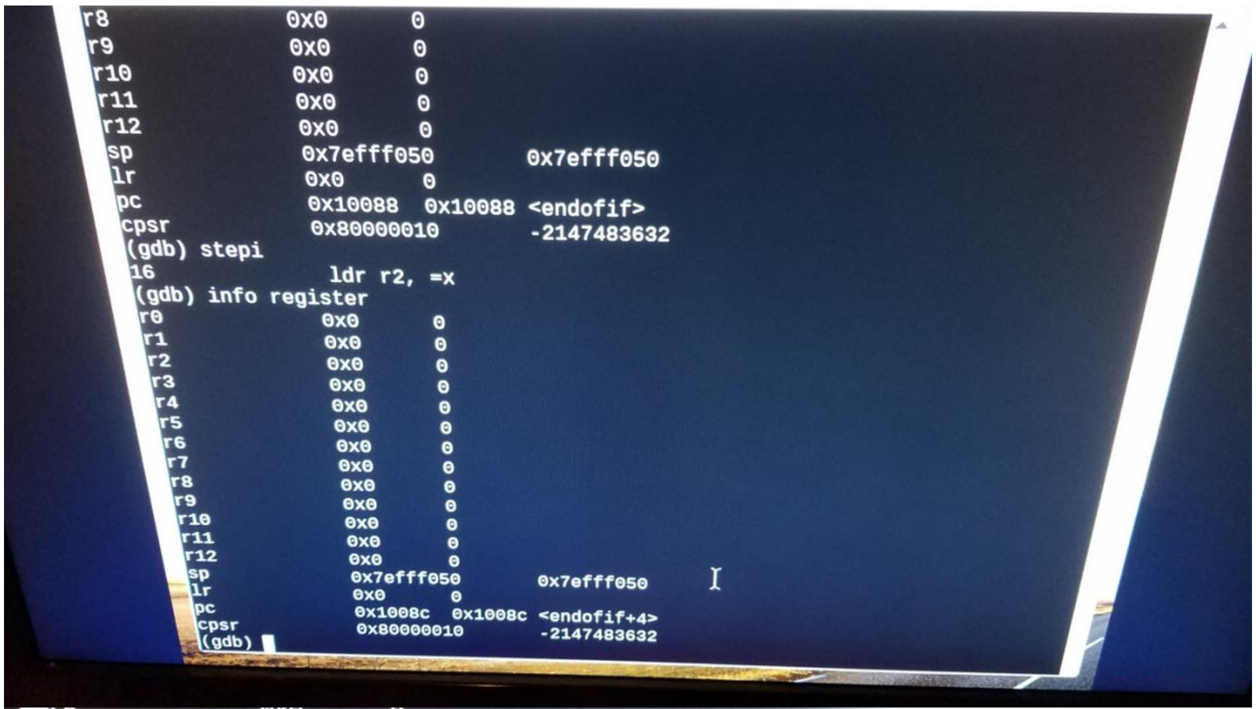


```
@ Control Structure 1 program

.section .data
x: .word 1
.section .text
.globl _start
_start:
        ldr r1, =x
        ldr r1, [r1]
        cmp r1, #3
        ble endofif
thenpart:
        sub r1, r1, #1
endofif:
        sub r1, r1, #1
        ldr r2, =x
        str r1, [r2]
        mov r7, #1
        svc #0
        .end
~
~
~
:x
```

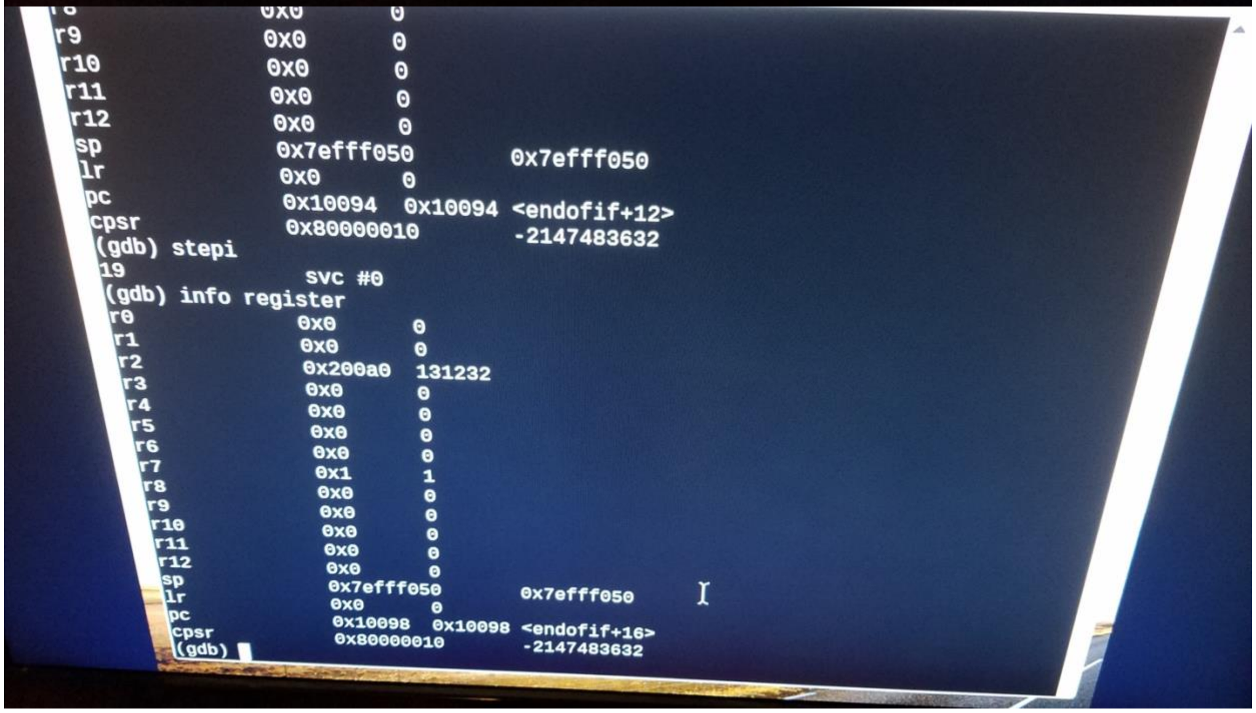
The 4 and 0 were simply used as test variables above to ensure the logic behind the code was working. In this example, I actually assign 1 to the variable x as the question states.
You can also see the cpsr flags (z flag) below, as well as the change in the x variable in its memory location (address: 131232 in this example).



```
r0             0x0      0
r1             0x0      0
r2             0x200a0  131232
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             0x7efff050       0x7efff050
lr             0x0      0
pc             0x10090  0x10090 <endofif+8>
cpsr           0x80000010       -2147483632
(gdb) x/1xw 131232
0x200a0:        0x00000001
(gdb) stepi
18              mov r7, #1
(gdb) x/1xw 131232
0x200a0:        0x00000000
(gdb) 
```



```
r8             0x0        0
r9             0x0        0
r10            0x0        0
r11            0x0        0
r12            0x0        0
sp             0x7efff050       0x7efff050
lr             0x0        0
pc             0x10088  0x10088 <endofif>
cpsr           0x80000010       -2147483632
(gdb) stepi
16              ldr r2, =x
(gdb) info register
r0             0x0        0
r1             0x0        0
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             0x7efff050       0x7efff050
lr             0x0        0
pc             0x1008c  0x1008c <endofif+4>
cpsr           0x80000010       -2147483632
(gdb)
```



```
r9             0x0        0
r10            0x0        0
r11            0x0        0
r12            0x0        0
sp             0x7efff050       0x7efff050
lr             0x0        0
pc             0x10094  0x10094 <endofif+12>
cpsr           0x80000010       -2147483632
(gdb) stepi
19              svc #0
(gdb) info register
r0             0x0        0
r1             0x0        0
r2             0x200a0    131232
r3             0x0        0
r4             0x0        0
r5             0x0        0
r6             0x0        0
r7             0x1        1
r8             0x0        0
r9             0x0        0
r10            0x0        0
r11            0x0        0
r12            0x0        0
sp             0x7efff050       0x7efff050
lr             0x0        0
pc             0x10098  0x10098 <endofif+16>
cpsr           0x80000010       -2147483632
(gdb)
```