Task 4

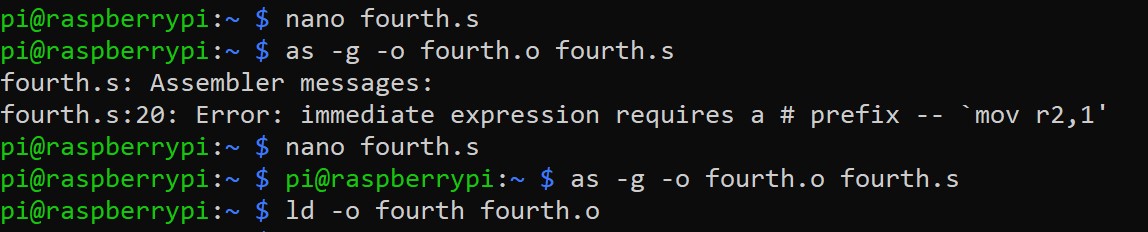
I begin by typing the fourth.s program in Raspberry PI using nano command (Figure 1).

A screenshot of a cell phone

Description automatically generated

*Figure 1 – fourth program*

While making a program an executable, I noticed an error. The sign “#” was not added in front of the immediate. I fixed the code and repeat my steps. This time, the code compiled, and I was able to run it (Figure 2).



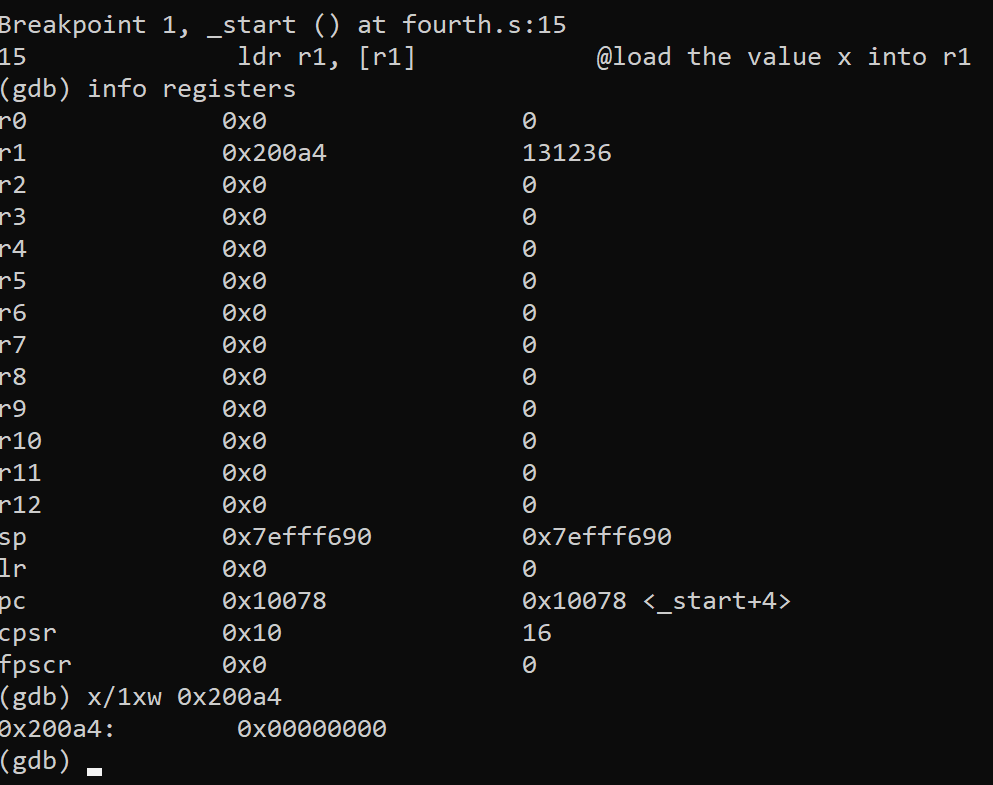
*Figure 2 – compiling fourth program*

Then, I began tracing the code using debugger (Figure 3).

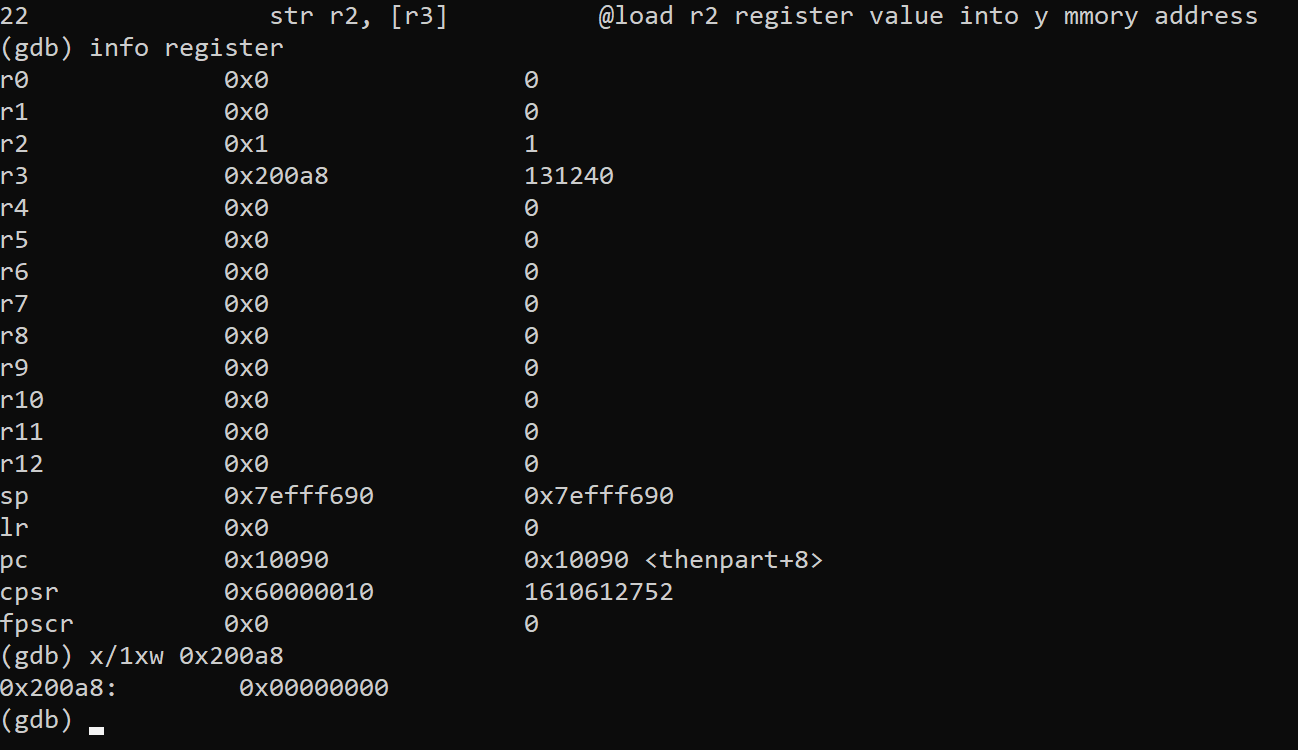
*A close up of text on a black background

Description automatically generated*

*Figure 3 – listing the code using debugger*

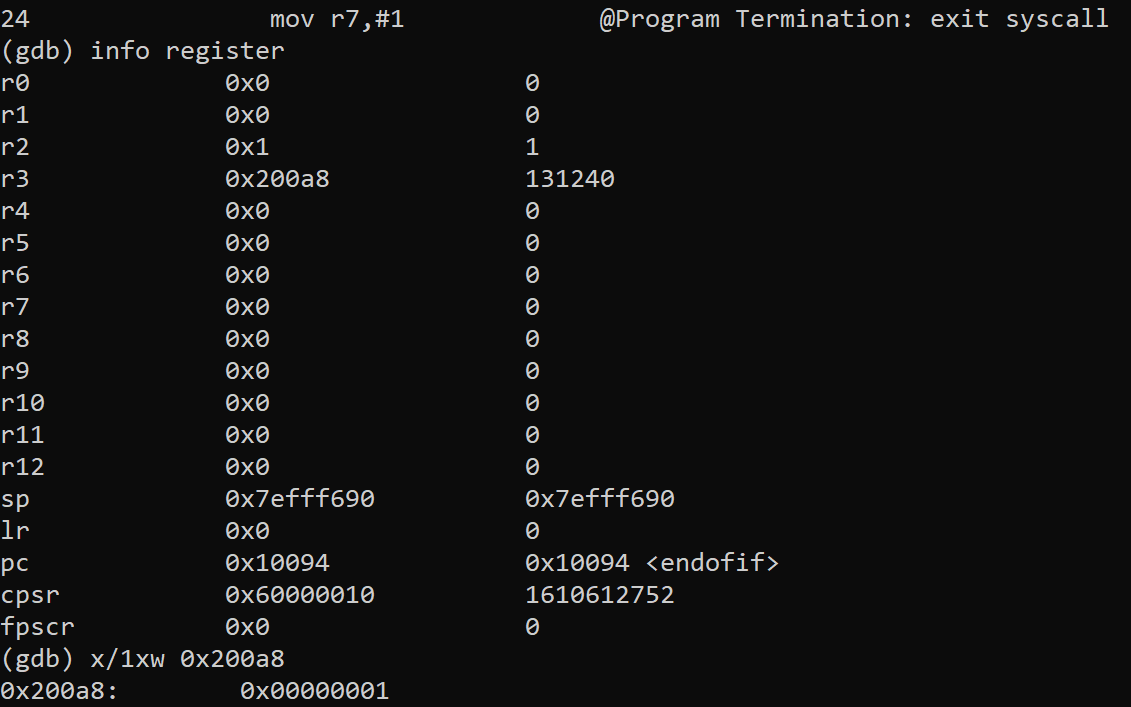
As I was debugging the program, I checked the memory addresses of both x (Figure 4) and y (Figure 5).

*Figure 4 – checking the memory address of x*

**

*Figure 5 – checking the memory address of y*

Then after running the entire program I checked the value of the memory address of r2 where I previously stored y, and now it shows the value 1 (Figure 7). This is to be expected since the program match the conditional statement(beg=branch jump when it is equal to zero) so instead of executing the next line(b endofif), it jumps to the procedure called thenpart.



*Figure 7 – debugging fourth program*

I’ve also used cpsr to check which flags the program set. The value that was shown is in 32 bits. The flags that were set are carry and zero flags (Figure 8). Since the expression subtracts equal numbers 0 from 0, it explains why these two flags were set.

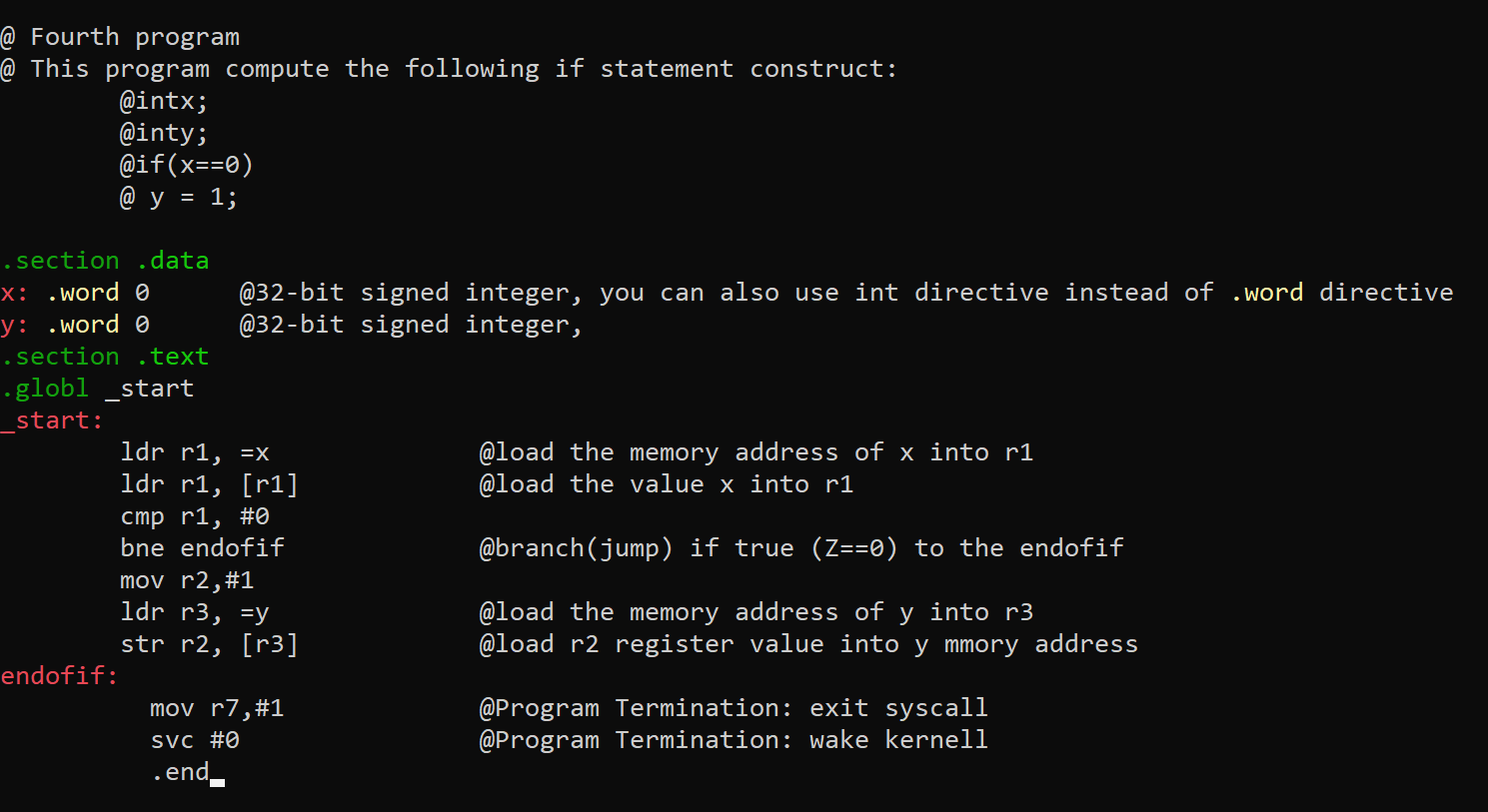
A close up of a logo

Description automatically generated

*Figure 8 – flags*

b)

I began by replacing beq with bne and removing b instructions from the given code from part a to make the program more efficient (Figure 1). This change will allow the program to only jump, when the numbers are not equal.



*Figure 1 – edited fourth program*

Then I began debugging the code using debugger. I set a breakpoint at line 15 and checked the registers.

After checking the memory, I noticed that the value of x on r1 is equal to 0 (Figure 2)

A screenshot of a cell phone

Description automatically generated

*Figure 2 – the value of x*

Then, I repeated the same steps for r2, to check the value of y (Figure 3).

A screenshot of a cell phone

Description automatically generated

*Figure 3 – the value of y*

After running the program, I noticed the result that I got was the same as in part a. The program didn’t jump to “endofif” and assigned the value 1 to y (Figure 4).

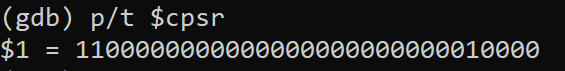
A screenshot of a cell phone

Description automatically generated

*Figure 4 – debugging the fourth program*

After running the program, I also checked which flags were raised (Figure 5). The result that I got was the same as in part a, where both zero and carry flags were set.

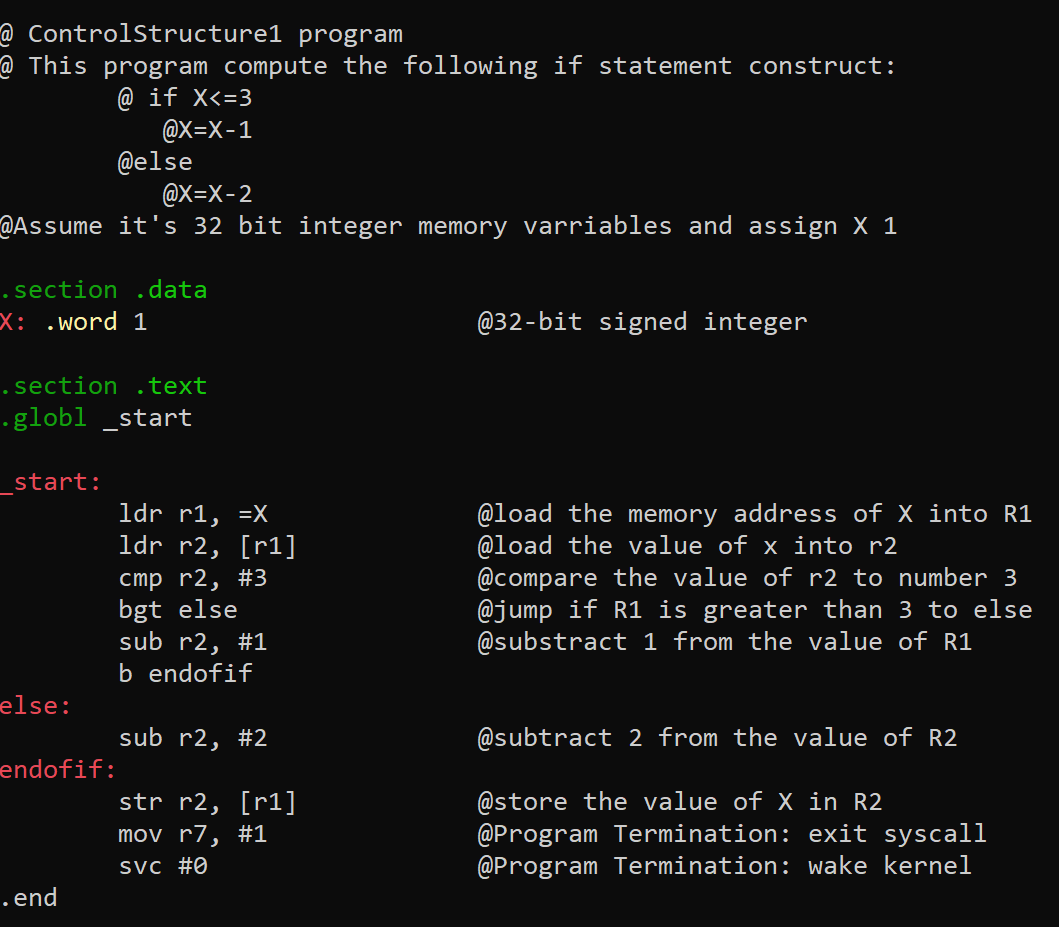
Flags for part B



*Figure 5 – checking the flags*

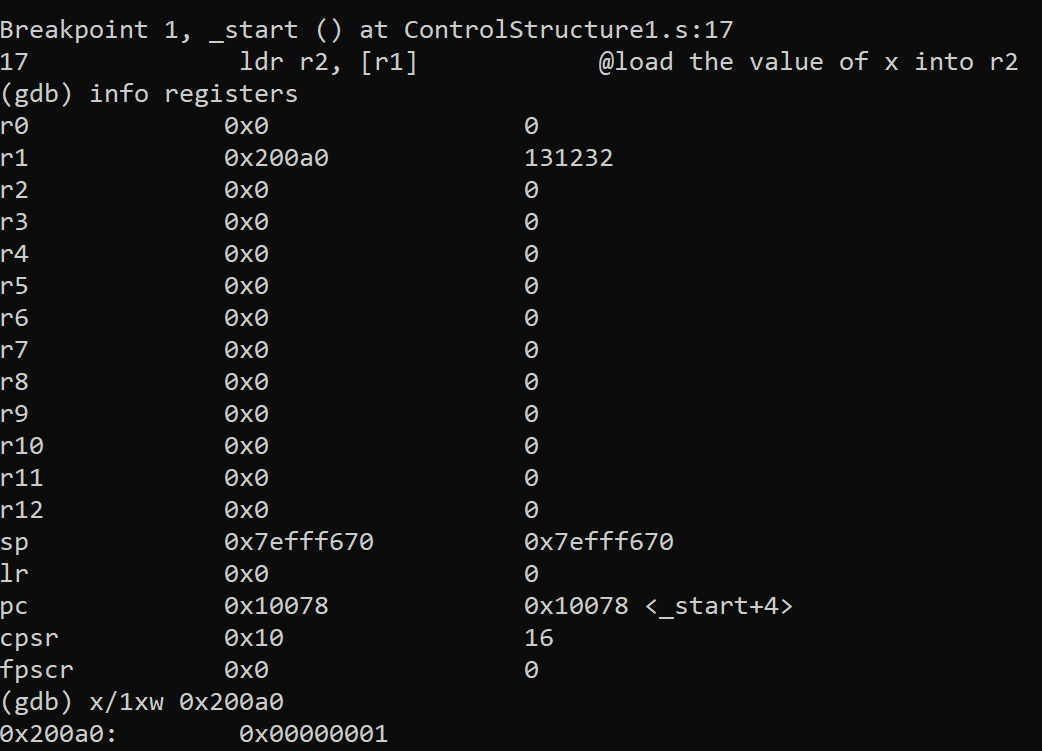
Part c)

I began by typing the code for the given expression (Figure 1).



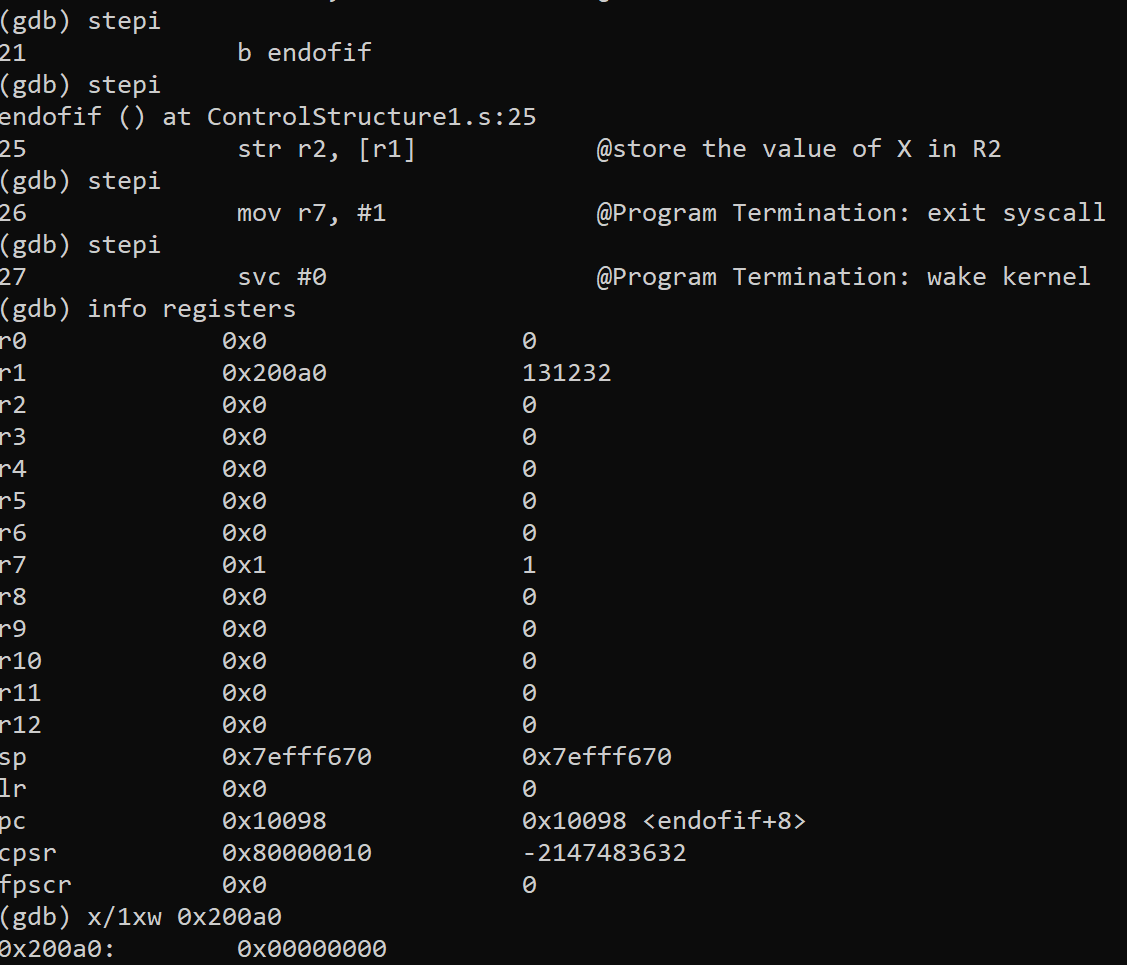
*Figure 1 – ControlStructure1 program*

Then I began debugging it. The value of X is not stored on R2 (Figure 2).

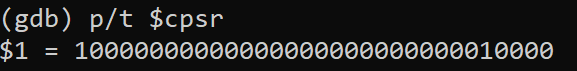


*Figure 2 – value of X stored on R2*

After running the code and checking the flags, I noticed that the negative flag is set (Figure 4). The negative flag is raised because 1 is less than 3 and by subtracting 3 from 1 I get negative number. Since I initialized X as 1, I’m subtracting 1-1, which results in 0 (Figure 3).

**

*Figure 3 – debugging the code*

**

*Figure 4 - flags*