

Week 7 Lab - Part 7.1

On the right side of the ARMLite simulator is a grid of memory addresses, with each block of 8 hex digits (all initialised to 0) representing a 32 bit word.

Click on any visible memory word and type in 101 (followed by the "Enter" key).

7.1.1 What value is displayed? Why?

- It displayed 0x00000065. Because typing 101, the simulator understood the inputted value as decimal value, hex representation of 101 is 0x00000065. The memory field displayed value in hex.

Click on another memory word, enter 0x101

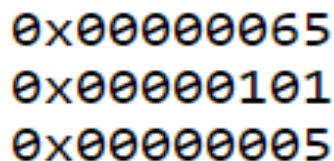
7.1.2 What value is displayed, and why?

- It displayed 0x00000101. Because Hex literals start with a 0x, entering 0x101 means we are inputting hexadecimal value of 0x101. The memory field displayed value in hex.

On another memory word, enter 0b101

7.1.3 What value is displayed, and why?

- It displayed 0x00000005. Because binary literals start with a 0b, entering 0b101 means we are inputting binary value of 101 (5 in decimal representation), Hex representation of 101 (5 in decimal representation) is 0x00000005. The memory field displayed value in hex.



0x00000065
0x00000101
0x00000005

If you now hover (don't click) the mouse over any of the memory words where you have entered a value you will get a pop-up 'tooltip'.

What does the tooltip tell you?

- The tool tips tell us the value in binary and decimal representation

Below the grid of memory words is a drop down menu that looks like this:



This drop-down selector allows you to change the base in which data is displayed. Changing the base does not change the underlying data value, only the base number system in which the value is displayed.

Change this to Decimal (unsigned) and note the change that has occurred to the three memory words you previously entered.

When you mouse over one of these words, what now appears in the tooltip?

7.1.4: Does changing the representation of the data in memory also change the representation of the row and column-headers (the white digits on a blue background)? Should it?

- Changing the representation of the data in memory doesn't change the representation of the row and column-headers . It should remain the same as we need those of access to the memory (memory addressing).

Week 7 Lab - Part 7.2

These values represent the first four digits of the address for all memory words in that row.

000
0x0000
0x0001
0x0002
0x0003
0x0004
0x0005
0x0006
0x0007
0x0008
0x0009
0x000a
0x000b
0x000c
0x000d
0x000e
0x000f
0x0010

These single digit hex values represent offsets from the row-header address.

0x0	0x4	0x8	0xc
0x00000000	0x00000000	0x00000000	0x00000000

==> Therefore, the full address of any memory word is obtained by appending the column header digit to the 4 digit row-header. For example, the address of the top-left word on this screen is 0x00000, and the bottom-right is 0x001fc

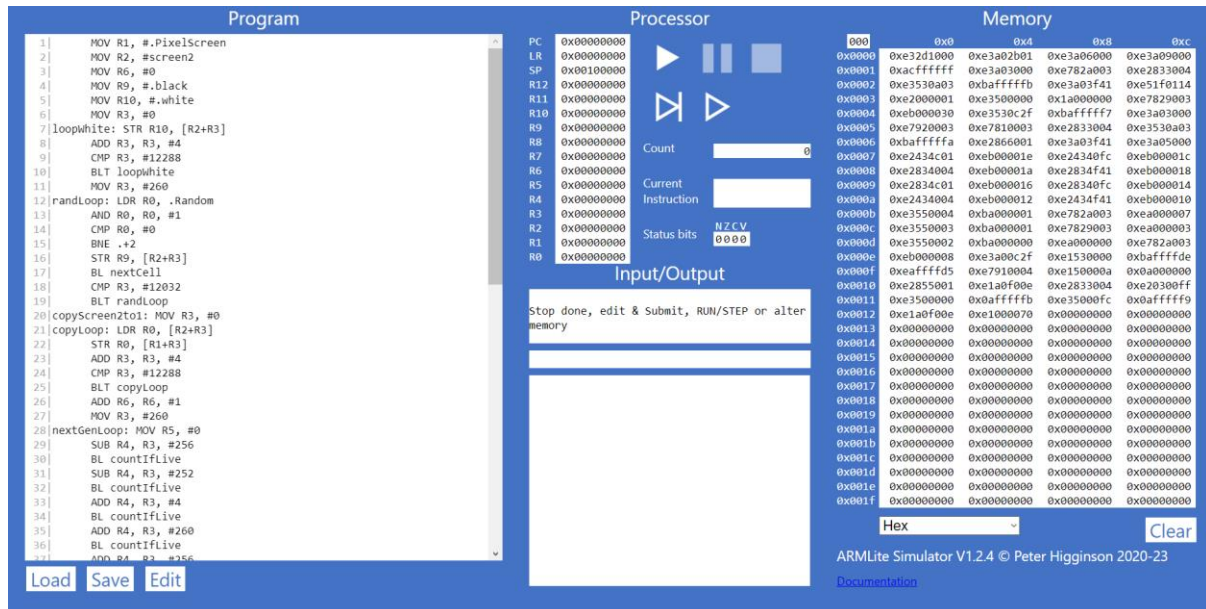
7.2.1 Notice these column header memory address offsets go up in multiples of 0x4. Why is this ?

Hint: Remember how many bits are in each memory word!

- Each memory word is 32 bits long and each hexadecimal digit represents 4 bits so the memory address offsets increase by multiples of 0x4

Week 7 Lab - Part 7.3

7.3.1 Take a screen shot of the simulator in full and add it to your submission document



7.3.2 Based on what we've learnt about assemblers and Von Neuman architectures, explain what you think just happened.

- The assemblers read the instruction of the sources code in the program window, storing them into memory, and await for execution.

Hover the mouse over one of the lines of the source code (after the code has been submitted).

You will see a pop-up tooltip showing a 5 digit hex value.

7.3.3 Based on what we have learnt about memory addressing in ARMLite, and your response to 7.3.2, what do you think this value represents ?

- The value represent the memory location (address) of where that line of code (instruction) is stored in the memory.

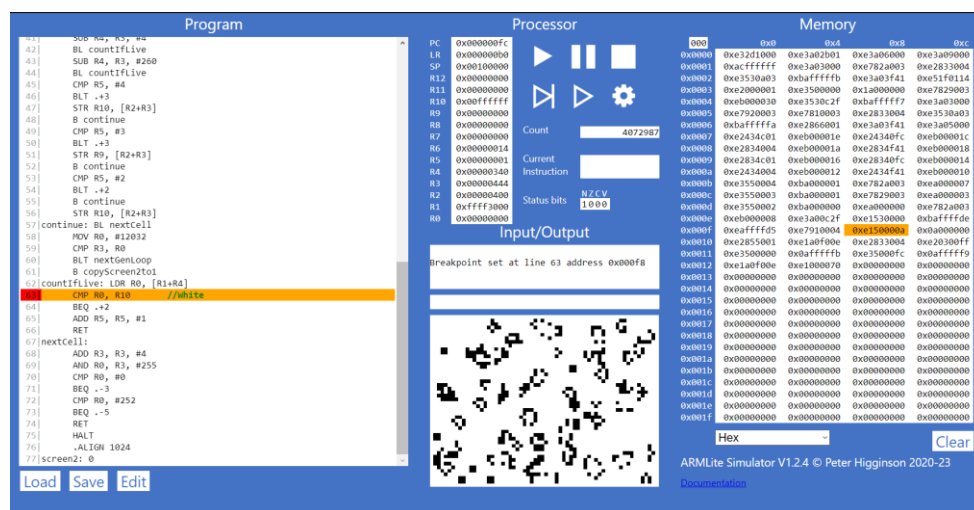
Run the program once more.

While the program is executing, click the **Pause** button (between the **Run** and **Stop** buttons).

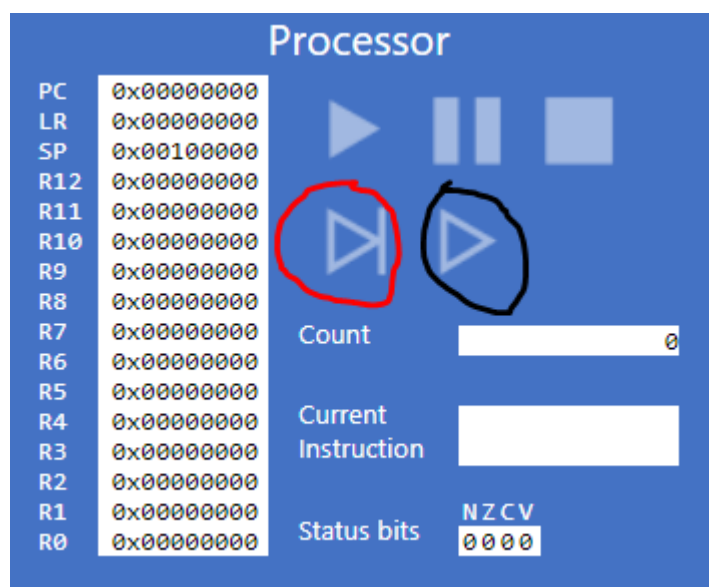
As well as freezing the graphics screen, you will also see orange highlighting appear in both the Program and Memory windows.

7.4.1 What do you think the highlighting in both windows signifies ?

- The highlighted line in the program window indicates the line of code was executed before the program was paused.
- The highlighted line in memory window indicates where that line of code is stored.



Now click the button circled in red below.

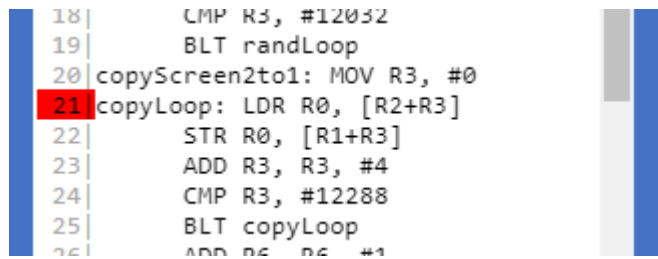


7.4.2 What do you think happens when you click the button circled in red ?

- The highlighted lines in both program and memory window jumped to the next line and the instruction is executed

Finally, while paused, click line number 21 of the source code in the Program Window.

This will paint a red background behind the line number like this:



The screenshot shows a list of assembly instructions in a program window. The instructions are numbered 18 through 25. Line 21, which contains the instruction 'copyLoop: LDR R0, [R2+R3]', is highlighted with a red background behind the line number. The instructions are as follows:

```
18|      CMP R3, #12032
19|      BLT randLoop
20|copyScreen2to1: MOV R3, #0
21|copyLoop: LDR R0, [R2+R3]
22|      STR R0, [R1+R3]
23|      ADD R3, R3, #4
24|      CMP R3, #12288
25|      BLT copyLoop
26|      ADD R5, R5, #1
```

This is called 'setting a breakpoint' and will cause processing to be paused when the breakpoint is reached.

Having set the breakpoint, continue running until the pause is observed (almost immediately!).

7.4.3 Has the processor paused just before, or just after executing the line with the breakpoint ?

- The processor paused just before executing the line with the breakpoint

Week 7 Lab - Part 7.5

7.5.1 Before executing this instruction, describe in words what you think this instruction is going to do, and what values you expect to see in R0 and R1 when it is complete ?

- First, the assembler will take the value saved in R0, which is 1, and add to 8. The result will be saved to R1.

7.5.2 When the program is complete, take a screen shot of the register table showing the values.

Program

```
1: MOV R1, #300
2: ADD R1, R1, #21
3: ADD R1, R1, #5
4: SUB R1, R1, #64
5: ADD R1, R1, #92
6: SUB R1, R1, #18
7: MOV R2, #254
8: SUB R2, R2, R1
9: ADD R7, R7, R1
10: HALT
```

Processor

PC: 0x00000028
LR: 0x00000000
SP: 0x00100000
R12: 0x00000000
R11: 0x00000000
R10: 0x00000000
R9: 0x00000000
R8: 0x00000000
R7: 0x00000000
R6: 0x00000000
R5: 0x00000000
R4: 0x00000000
R3: 0x00000000
R2: 0x00000000
R1: 0x00000000
R0: 0x00000000

Count: 10
Current Instruction:
Status bits: NZCV 0000

Input/Output

Breakpoint removed at line 2 address 0x000004

Memory

000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x0001 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x0002 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x0003 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x0004 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x0005 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x0006 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x0007 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x0008 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x0009 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x000a 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x000b 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x000c 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x000d 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x000e 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x000f 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x0010 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x0011 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x0012 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x0013 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x0014 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x0015 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x0016 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x0017 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x0018 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x0019 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x001a 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x001b 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x001c 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x001d 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x001e 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x001f 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000

ARMLite Simulator V1.2.4 © Peter Higginson 2020-23
[Documentation](#)

7.5.3 Task: Your 6 initial numbers are now 300, 21, 5, 64, 92, 18. Write an Assembly Program that uses these values to compute a final value of 294 (you need only use MOV, ADD and SUB). Place your final result in register R7 (don't forget the HALT instruction)

Program

```
1: MOV R0, #1
2: ADD R1, R0, #8
3: ADD R2, R1, #100
4: SUB R3, R2, #25
5: HALT
```

Processor

PC: 0x00000008
LR: 0x00000000
SP: 0x00100000
R12: 0x00000000
R11: 0x00000000
R10: 0x00000000
R9: 0x00000000
R8: 0x00000000
R7: 0x00000000
R6: 0x00000000
R5: 0x00000000
R4: 0x00000000
R3: 0x00000000
R2: 0x00000000
R1: 0x00000000
R0: 0x00000001

Count: 2
Current Instruction: ADD R1, R0, #8
Status bits: NZCV 0000

Input/Output

Done instruction ADD R1, R0, #8 at line 2

Memory

000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x0001 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x0002 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x0003 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x0004 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x0005 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x0006 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x0007 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x0008 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x0009 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x000a 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x000b 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x000c 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x000d 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x000e 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x000f 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x0010 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x0011 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x0012 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x0013 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x0014 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x0015 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x0016 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x0017 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x0018 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x0019 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x001a 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x001b 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x001c 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x001d 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x001e 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x001f 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000

ARMLite Simulator V1.2.4 © Peter Higginson 2020-23
[Documentation](#)

7.5.4 Task: Write your own simple program, that starts with a MOV (as in the previous example) followed by five instructions, using each of the five new instructions listed above, once only, but in any order you like – plus a HALT at the end, and with whatever immediate values you like.

Enter your program into ARMLite, submit the code and when its ready to run, step through the program, completing the table below (make a copy of it in your submission document)

Instruction	Decimal value of the destination register after executing this instruction	Binary value of the destination register after executing this instruction
MOV R1, #10	10	1010
AND R2, R1, #4	0	0
ORR R3, R1, #5	15	1111
EOR R4, R1, #15	5	101
LSL R5, R1, #3	80	1010000
LSR R6, R1, #20	0	0

Task 7.5.5 Lets play the game we played in 7.5.3, but this time you can use any of the instructions listed in this lab so far (ie., MOV, AND, OR, and any of the bit-wise operators).

Your six initial numbers are: 12, 11, 7, 5, 3, 2 and your target number is: 79

The screenshot shows the ARMLite Simulator V1.2.4 interface. The 'Program' window contains the following assembly code:

```

1| MOV R0, #12
2| MOV R1, #11
3| MOV R2, #7
4| MOV R3, #5
5| MOV R4, #3
6| MOV R5, #2
7| ADD R6, R0, R1
8| MOV R7, R6
9| ADD R6, R0, R6
10| ADD R6, R6, R7
11| ADD R8, R2, R4
12| SUB R9, R4, R5
13| ADD R6, R6, R8
14| ADD R6, R6, R8
15| ADD R6, R6, R9
16| HALT

```

The 'Processor' window shows the following registers and values:

Register	Value
PC	0
LR	0
SP	1048576
R12	0
R11	0
R10	0
R9	0
R8	0
R7	0
R6	0
R5	0
R4	0
R3	0
R2	0
R1	0
R0	0

The 'Memory' window shows a table of memory addresses and values:

Address	Value
0x0000	3818913804
0x0001	3818930179
0x0002	3766509574
0x0003	3766902792
0x0004	0
0x0005	0
0x0006	0
0x0007	0
0x0008	0
0x0009	0
0x000a	0
0x000b	0
0x000c	0
0x000d	0
0x000e	0
0x000f	0
0x0010	0
0x0011	0
0x0012	0
0x0013	0
0x0014	0
0x0015	0
0x0016	0
0x0017	0
0x0018	0
0x0019	0
0x001a	0
0x001b	0
0x001c	0
0x001d	0
0x001e	0
0x001f	0

The 'Input/Output' window shows 'Program assembled. Run or Step to execute'.

Instruction	Decimal value of the destination register after executing this instruction	Binary value of the destination register after executing this instruction
MOV R0, #12	12	1100
MOV R1, #11	11	1011
MOV R2, #7	7	111
MOV R3, #5	5	101
MOV R4, #3	3	11
MOV R5, #2	2	10
ADD R6, R0, R1	23	10111
MOV R7, R6	23	10111
ADD R6, R0, R6	35	100011

ADD R6, R6, R7	58	111010
ADD R8, R2, R4	10	1010
SUB R9, R4, R5	1	0001
ADD R6, R6, R8	68	1000100
ADD R6, R6, R8	78	1001110
ADD R6, R6, R9	79	1001111

Task 7.5.6: Let's play again !

Your six initial numbers are: 99, 77, 33, 31, 14, 12 and your target number is: 32

The screenshot displays the ARMLite Simulator V1.2.4 interface, which is divided into three main sections: Program, Processor, and Memory.

- Program Section:** Contains a list of assembly instructions:


```

1| MOV R0, #77
2| SUB R0, R0, #33
3| SUB R0, R0, #12
4| HALT
      
```
- Processor Section:** Shows the state of the processor registers and control elements:
 - Registers:** PC (0x00000010), LR (0x00000000), SP (0x00100000), R12 (0x00000000), R11 (0x00000000), R10 (0x00000000), R9 (0x00000000), R8 (0x00000000), R7 (0x00000000), R6 (0x00000000), R5 (0x00000000), R4 (0x00000000), R3 (0x00000000), R2 (0x00000000), R1 (0x00000000), R0 (0x00000020).
 - Control Elements:** Count (4), Current Instruction, Status bits (NZCV: 0000).
 - Input/Output:** A text box showing "ESC pressed to abort input".
- Memory Section:** Displays a memory dump with addresses from 0x00000000 to 0x0000001f. The value at address 0x00000000 is 0xe3a0004d, and the value at address 0x00000001 is 0xe2400021. The value at address 0x00000002 is 0xe1000070, which is highlighted in orange.

At the bottom of the interface, there are buttons for "Load", "Save", "Submit", and "Revert". The footer text reads: "ARMLite Simulator V1.2.4 © Peter Higginson 2020-23" and includes a link to "Documentation".

Instruction	Decimal value of the destination register after executing this instruction	Binary value of the destination register after executing this instruction
MOV R0, #77	77	1001101
SUB R0, R0, #33	44	101100
SUB R0, R0, #12	32	100000

Week 7 Lab - Part 7.6

Copy and Paste the following code into the ARMLite code editor and submit the code.

```
MOV R0, #9999
LSL R1, R0, #18
HALT
```

Before executing, switch ARMLite to display data in memory in *Decimal (signed)* using the drop down box below the memory grid.

Now run the program and note the result in register R1.

7.6.1 - Why is the result shown in R1 a negative decimal number, and with no obvious relationship to 9999 ?

- Because due to the 2's complement representation and the last signed bits after shifting the 9999 left by 18 bits, turning this signed bit into 1 (which represent a negative number).

7.6.3 - What is the binary representation of each of these signed decimal numbers: 1, -1, 2, -2

What pattern do you notice ? Make a note of these in your submission document before reading on.

Signed decimal	Binary Representation
1	0001
-1	11111111111111111111111111111111
2	0010
-2	11111111111111111111111111111110

7.6.4 - Write an ARM Assembly program that converts a positive decimal integer into its negative version. Start by moving the input value into R0, and leaving the result in R1.

The screenshot displays the ARMLite Simulator V1.2.4 interface, which is used for simulating ARM assembly programs. The interface is divided into several sections:

- Program:** A list of assembly instructions. The current instruction being executed is `ADD R1, R1, #1` at address 0x00000003.
- Processor:** A section showing the state of the processor registers and control flags. The Program Counter (PC) is 16, and the Status bits are NZCV (0000). The Count register is set to 4.
- Memory:** A table showing the memory contents. The address 0x00000003 contains the value -520093584, which is the negative version of the input value 520093584.
- Input/Output:** A section for monitoring the program's input and output. The status bar indicates "Program HALTED. STOP, LOAD or EDIT".

The program being simulated is as follows:

```

1| MOV R0, #3
2| MOV R1, R0
3| ADD R1, R1, #1
4| HALT
  
```

The memory dump shows the following values:

Address	Value
0x00000000	-476053501
0x00000001	0
0x00000002	0
0x00000003	-520093584
0x00000004	0
0x00000005	0
0x00000006	0
0x00000007	0
0x00000008	0
0x00000009	0
0x0000000a	0
0x0000000b	0
0x0000000c	0
0x0000000d	0
0x0000000e	0
0x0000000f	0
0x00000010	0
0x00000011	0
0x00000012	0
0x00000013	0
0x00000014	0
0x00000015	0
0x00000016	0
0x00000017	0
0x00000018	0
0x00000019	0
0x0000001a	0
0x0000001b	0
0x0000001c	0
0x0000001d	0
0x0000001e	0
0x0000001f	0

The simulator is running on a Windows operating system, and the ARMLite Simulator V1.2.4 logo is visible in the bottom right corner.