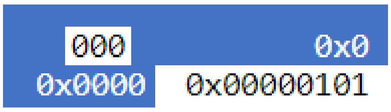
Name: Hridoy Ahmed

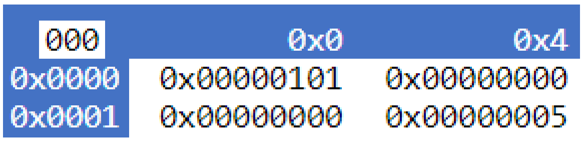
Student ID: 103798793

7.1.1 What value is displayed ? Why ?



```
   000          0x0
0x0000   0x00000101
```

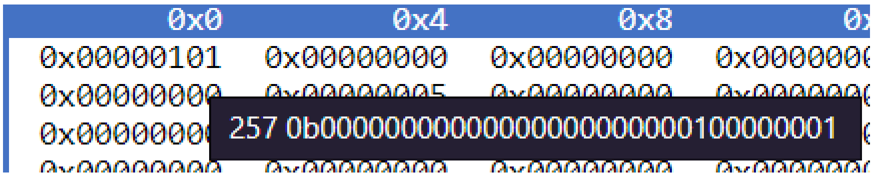
7.1.2 What value is displayed, and why?



```
 000          0x0          0x4
0x0000  0x00000101   0x00000000
0x0001  0x00000000   0x00000005
```

7.1.3 What value is displayed, and why?

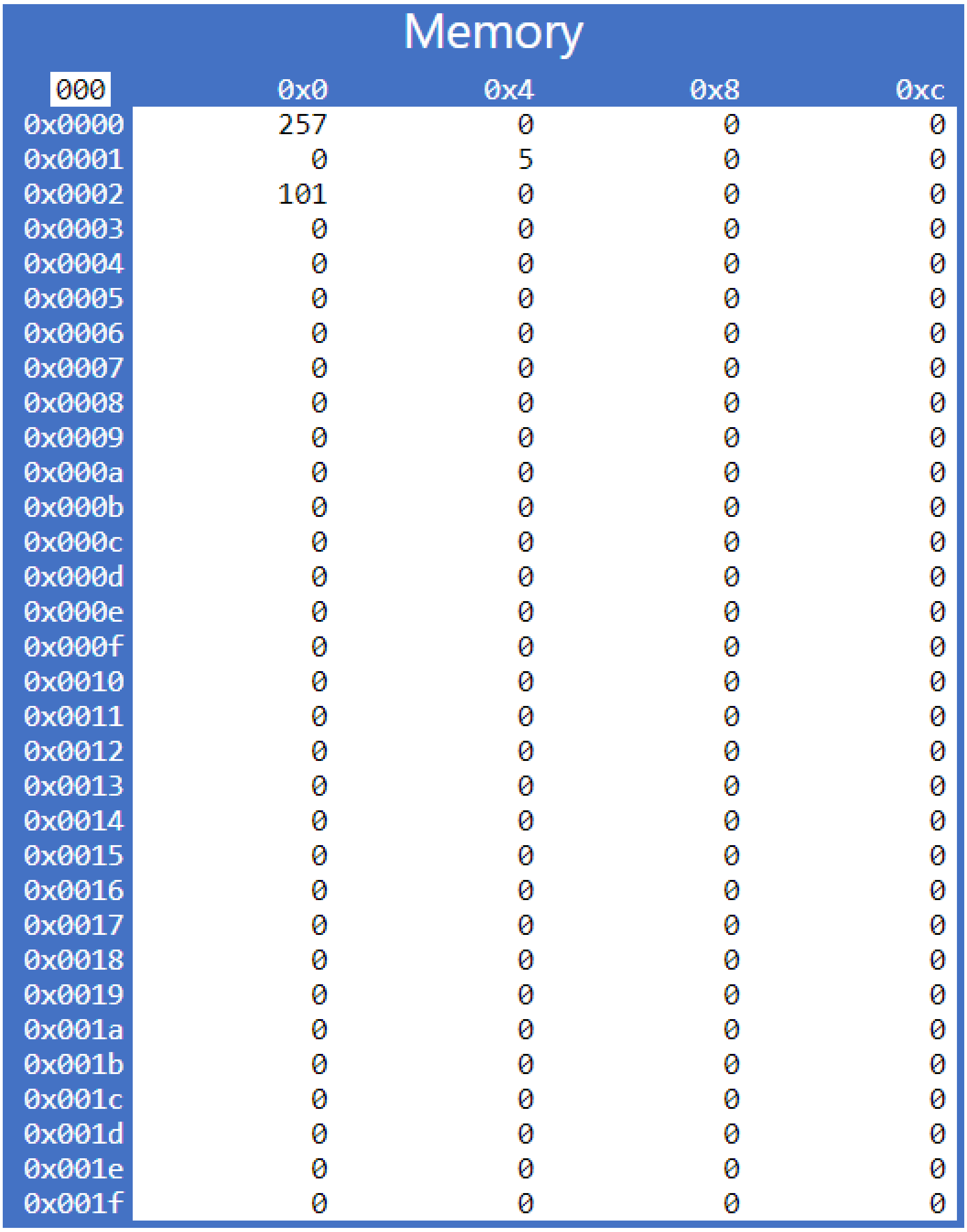
When I hover my cursor over the entered value, a tooltip displayed:



```
          0x0          0x4          0x8          0x
  0x00000101   0x00000000   0x00000000   0x0000000
  0x00000000   0x00000005   0x00000000   0x0000000
  0x0000000 257 0b00000000000000000000000100000001
```

Which is the binary representation of the entered value and its value in decimal.

Changing the grid display to Decimal (unsigned) gave us:



Memory

000	0x0	0x4	0x8	0xc
0x0000	257	0	0	0
0x0001	0	5	0	0
0x0002	101	0	0	0
0x0003	0	0	0	0
0x0004	0	0	0	0
0x0005	0	0	0	0
0x0006	0	0	0	0
0x0007	0	0	0	0
0x0008	0	0	0	0
0x0009	0	0	0	0
0x000a	0	0	0	0
0x000b	0	0	0	0
0x000c	0	0	0	0
0x000d	0	0	0	0
0x000e	0	0	0	0
0x000f	0	0	0	0
0x0010	0	0	0	0
0x0011	0	0	0	0
0x0012	0	0	0	0
0x0013	0	0	0	0
0x0014	0	0	0	0
0x0015	0	0	0	0
0x0016	0	0	0	0
0x0017	0	0	0	0
0x0018	0	0	0	0
0x0019	0	0	0	0
0x001a	0	0	0	0
0x001b	0	0	0	0
0x001c	0	0	0	0
0x001d	0	0	0	0
0x001e	0	0	0	0
0x001f	0	0	0	0

And when you hover your cursor over any of the three previously entered values, it will display the its value in hex and binary.

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 ?

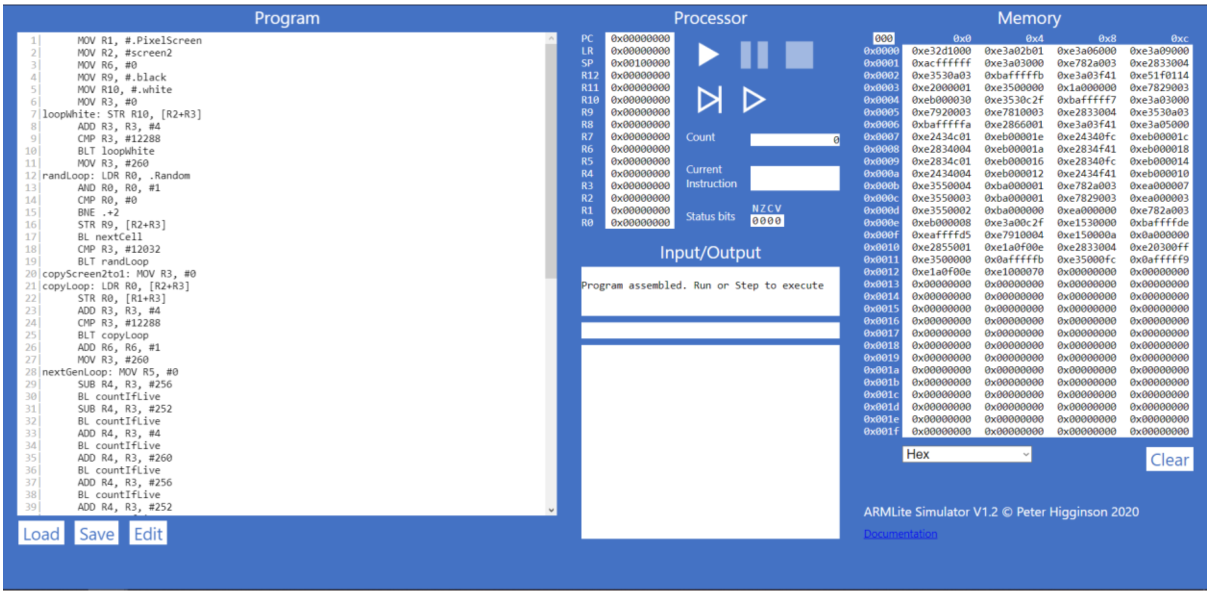
When changing the representation of the data from hex to decimal (unsigned), the row and column-headers remains.

I think it shouldn't as the row and column-headers are more like location for storing value or data, changing them wouldn't change where you wouldn't store your value or data.

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

Each block in ARMLite is represented by 8 hex digits (all initialised to 0) representing a 32 bit word. To represent all 32 bits, you need 4 blocks, with 8 hex digits each, so the offset is 0x4.

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



```
      MOV R1, #.PixelScreen
      MOV R2, #screen2
      MOV R6, #0
      MOV R9, #.black
      MOV R10, #.white
      MOV R3, #0
loopWhite: STR R10, [R2+R3]
      ADD R3, R3, #4
      CMP R3, #12288
      BLT loopWhite
      MOV R3, #260
randLoop: LDR R0, .Random
      AND R0, R0, #1
      CMP R0, #0
      BNE .+2
      STR R9, [R2+R3]
      BL nextCell
      CMP R3, #12032
      BLT randLoop
copyScreen2to1: MOV R3, #0
copyLoop: LDR R0, [R2+R3]
      STR R0, [R1+R3]
      ADD R3, R3, #4
      CMP R3, #12288
      BLT copyLoop
      ADD R6, R6, #1
      MOV R3, #260
nextGenLoop: MOV R5, #0
      SUB R4, R3, #256
      BL countIfLive
      SUB R4, R3, #252
      BL countIfLive
      ADD R4, R3, #4
      BL countIfLive
      ADD R4, R3, #260
      BL countIfLive
      ADD R4, R3, #256
      BL countIfLive
      ADD R4, R3, #252
```

```
PC   0x00000000
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 0

Status bits NZCV 0000

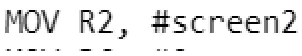
Program assembled. Run or Step to execute

000	0x0	0x4	0x8	0xc
0x0000	0xe32d1000	0xe3a02b01	0xe3a06000	0xe3a09000
0x0001	0xacffffff	0xe3a03000	0xe782a003	0xe2833004
0x0002	0xe3530a03	0xbafffffb	0xe3a03f41	0xe51f0114
0x0003	0xe2000001	0xe3500000	0x1a000000	0xe7829003
0x0004	0xeb000030	0xe3530c2f	0xbafffff7	0xe3a03000
0x0005	0xe7920003	0xe7810003	0xe2833004	0xe3530a03
0x0006	0xbafffffa	0xe2866001	0xe3a03f41	0xe3a05000
0x0007	0xe2434c01	0xeb00001e	0xe24340fc	0xeb00001c
0x0008	0xe2834004	0xeb00001a	0xe2834f41	0xeb000018
0x0009	0xe2834c01	0xeb000016	0xe28340fc	0xeb000014
0x000a	0xe2434004	0xeb000012	0xe2434f41	0xeb000010
0x000b	0xe3550004	0xba000001	0xe782a003	0xea000007
0x000c	0xe3550003	0xba000001	0xe7829003	0xea000003
0x000d	0xe3550002	0xba000000	0xea000000	0xe782a003
0x000e	0xeb000008	0xe3a00c2f	0xe1530000	0xbaffffde
0x000f	0xeaffffd5	0xe7910004	0xe150000a	0x0a000000
0x0010	0xe2855001	0xe1a0f00e	0xe2833004	0xe20300ff
0x0011	0xe3500000	0x0afffffb	0xe35000fc	0x0afffff9
0x0012	0xe1a0f00e	0xe1000070	0x00000000	0x00000000
0x0013	0x00000000	0x00000000	0x00000000	0x00000000
0x0014	0x00000000	0x00000000	0x00000000	0x00000000
0x0015	0x00000000	0x00000000	0x00000000	0x00000000
0x0016	0x00000000	0x00000000	0x00000000	0x00000000
0x0017	0x00000000	0x00000000	0x00000000	0x00000000
0x0018	0x00000000	0x00000000	0x00000000	0x00000000
0x0019	0x00000000	0x00000000	0x00000000	0x00000000
0x001a	0x00000000	0x00000000	0x00000000	0x00000000
0x001b	0x00000000	0x00000000	0x00000000	0x00000000
0x001c	0x00000000	0x00000000	0x00000000	0x00000000
0x001d	0x00000000	0x00000000	0x00000000	0x00000000
0x001e	0x00000000	0x00000000	0x00000000	0x00000000
0x001f	0x00000000	0x00000000	0x00000000	0x00000000

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

ARMlite take the source code that we just submitted and compiled it into Machine code. Then the simulated hardware in ARMlite read those Machine code and turn them into components to be put into the Memory section.

The 5 digits hex that displayed when we hover our cursor over. For instance, when hovering over.



```
MOV R2, #screen2
```

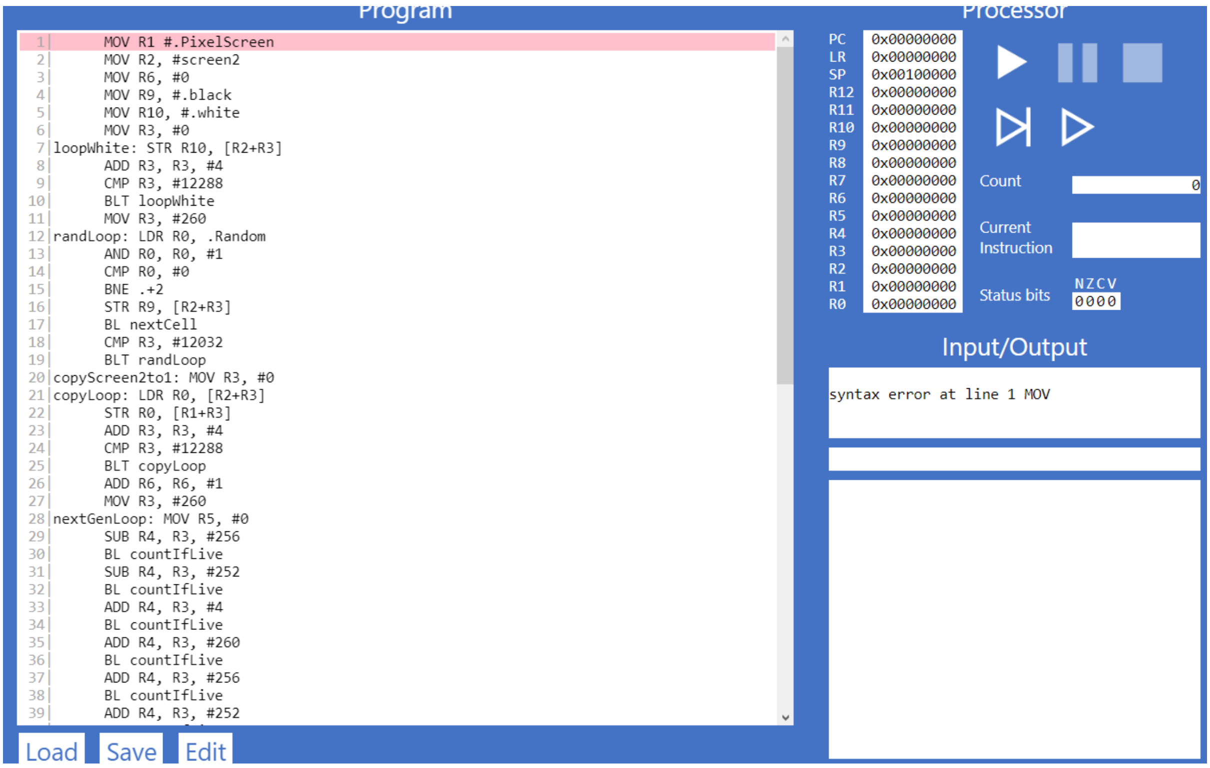
: it displayed 0x00004. Which is the value we move into R2.

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 ?

What has happened to them when hitting submit:

- The blank lines: they are removed
- Additional spaces: they are removed
- The comments: they will be highlighted in green
- The line numbers: they will disappeared
- The total number of instructions that end up as words in memory? (Why?). 74, because the other three instructions are have no values.

Removing the comma from the first line in the source code will prompt an error message.



```
        MOV R1 #.PixelScreen
        MOV R2, #screen2
        MOV R6, #0
        MOV R9, #.black
        MOV R10, #.white
        MOV R3, #0
loopWhite: STR R10, [R2+R3]
        ADD R3, R3, #4
        CMP R3, #12288
        BLT loopWhite
        MOV R3, #260
randLoop: LDR R0, .Random
        AND R0, R0, #1
        CMP R0, #0
        BNE .+2
        STR R9, [R2+R3]
        BL nextCell
        CMP R3, #12032
        BLT randLoop
copyScreen2to1: MOV R3, #0
copyLoop: LDR R0, [R2+R3]
        STR R0, [R1+R3]
        ADD R3, R3, #4
        CMP R3, #12288
        BLT copyLoop
        ADD R6, R6, #1
        MOV R3, #260
nextGenLoop: MOV R5, #0
        SUB R4, R3, #256
        BL countIfLive
        SUB R4, R3, #252
        BL countIfLive
        ADD R4, R3, #4
        BL countIfLive
        ADD R4, R3, #260
        BL countIfLive
        ADD R4, R3, #256
        BL countIfLive
        ADD R4, R3, #252
```

```
PC   0x00000000
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
```

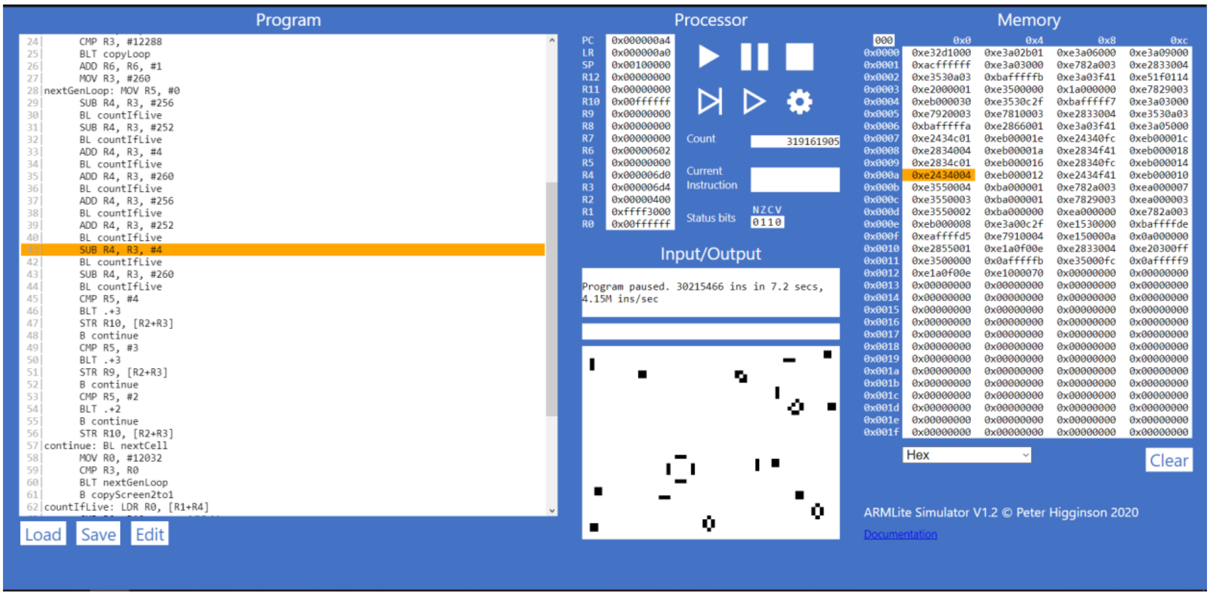
Count 0

Current Instruction

Status bits NZCV 0000

```
syntax error at line 1 MOV
```

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



```
        CMP R3, #12288
        BLT copyLoop
        ADD R6, R6, #1
        MOV R3, #260
nextGenLoop: MOV R5, #0
        SUB R4, R3, #256
        BL countIfLive
        SUB R4, R3, #252
        BL countIfLive
        ADD R4, R3, #4
        BL countIfLive
        ADD R4, R3, #260
        BL countIfLive
        ADD R4, R3, #256
        BL countIfLive
        ADD R4, R3, #252
        BL countIfLive
        SUB R4, R3, #4
        BL countIfLive
        SUB R4, R3, #260
        BL countIfLive
        CMP R5, #4
        BLT .+3
        STR R10, [R2+R3]
        B continue
        CMP R5, #3
        BLT .+3
        STR R9, [R2+R3]
        B continue
        CMP R5, #2
        BLT .+2
        B continue
        STR R10, [R2+R3]
continue: BL nextCell
        MOV R0, #12032
        CMP R3, R0
        BLT nextGenLoop
        B copyScreen2to1
countIfLive: LDR R0, [R1+R4]
```

```
PC   0x000000a4
LR   0x000000a0
SP   0x00100000
R12  0x00000000
R11  0x00000000
R10  0x00ffffff
R9   0x00000000
R8   0x00000000
R7   0x00000000
R6   0x00000602
R5   0x00000000
R4   0x000006d0
R3   0x000006d4
R2   0x00000400
R1   0xffff3000
R0   0x00ffffff
```

Count 319161905

Current Instruction

Status bits NZCV 0110

```
Program paused. 30215466 ins in 7.2 secs,
4.15M ins/sec
```

```
000            0x0         0x4         0x8         0xc
0x0000  0xe32d1000  0xe3a02b01  0xe3a06000  0xe3a09000
0x0001  0xacffffff  0xe3a03000  0xe782a003  0xe2833004
0x0002  0xe3530a03  0xbafffffb  0xe3a03f41  0xe51f0114
0x0003  0xe2000001  0xe3500000  0x1a000000  0xe7829003
0x0004  0xeb000030  0xe3530c2f  0xbafffff7  0xe3a03000
0x0005  0xe7920003  0xe7810003  0xe2833004  0xe3530a03
0x0006  0xbafffffa  0xe2866001  0xe3a03f41  0xe3a05000
0x0007  0xe2434c01  0xeb00001e  0xe24340fc  0xeb00001c
0x0008  0xe2834004  0xeb00001a  0xe2834f41  0xeb000018
0x0009  0xe2834c01  0xeb000016  0xe28340fc  0xeb000014
0x000a  0xe2434004  0xeb000012  0xe2434f41  0xeb000010
0x000b  0xe3550004  0xba000001  0xe782a003  0xea000007
0x000c  0xe3550003  0xba000001  0xe7829003  0xea000003
0x000d  0xe3550002  0xba000000  0xea000000  0xe782a003
0x000e  0xeb000008  0xe3a00c2f  0xe1530000  0xbaffffde
0x000f  0xeaffffd5  0xe7910004  0xe150000a  0x0a000000
0x0010  0xe2855001  0xe1a0f00e  0xe2833004  0xe20300ff
0x0011  0xe3500000  0x0afffffb  0xe35000fc  0x0afffff9
0x0012  0xe1a0f00e  0xe1000070  0x00000000  0x00000000
0x0013  0x00000000  0x00000000  0x00000000  0x00000000
0x0014  0x00000000  0x00000000  0x00000000  0x00000000
0x0015  0x00000000  0x00000000  0x00000000  0x00000000
0x0016  0x00000000  0x00000000  0x00000000  0x00000000
0x0017  0x00000000  0x00000000  0x00000000  0x00000000
0x0018  0x00000000  0x00000000  0x00000000  0x00000000
0x0019  0x00000000  0x00000000  0x00000000  0x00000000
0x001a  0x00000000  0x00000000  0x00000000  0x00000000
0x001b  0x00000000  0x00000000  0x00000000  0x00000000
0x001c  0x00000000  0x00000000  0x00000000  0x00000000
0x001d  0x00000000  0x00000000  0x00000000  0x00000000
0x001e  0x00000000  0x00000000  0x00000000  0x00000000
0x001f  0x00000000  0x00000000  0x00000000  0x00000000
```

Hex

ARMLite Simulator V1.2 © Peter Higginson 2020

The highlighted part on both windows signifies where the program is paused.

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

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

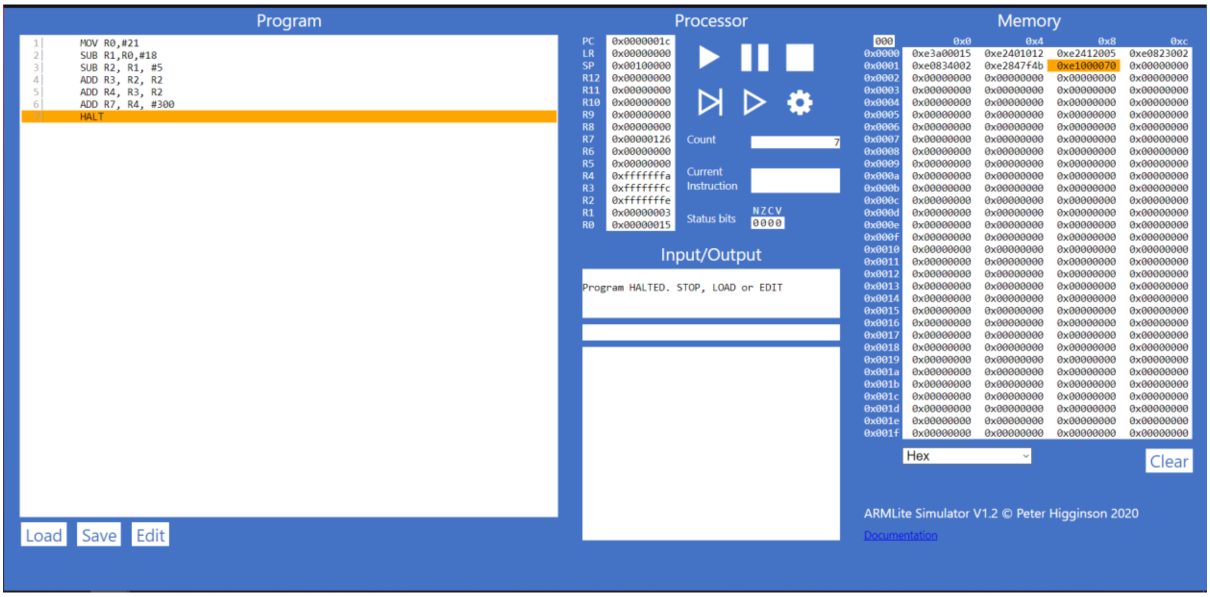
7.5.1 Before executing this instruction, describe in words what you think this instruction is going to do, and what values you expect to see in R0 and R1 when it is complete ?

```
MOV R0,#1  //Take 1 and move it to register 0
ADD R1,R0,#8 //Add 8 and the value in R0 together then store it in R
1
ADD R2,R1,#100 //Add 100 and the value in R1 together then store it
in R2
SUB R3,R2,#25 //Subtract 25 in R2 then store it in R3
HALT //Tell the program to stop
```

The program complete when it hit execute halt.

***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)**

When the program is complete, take a screen shot of the code and the register table.



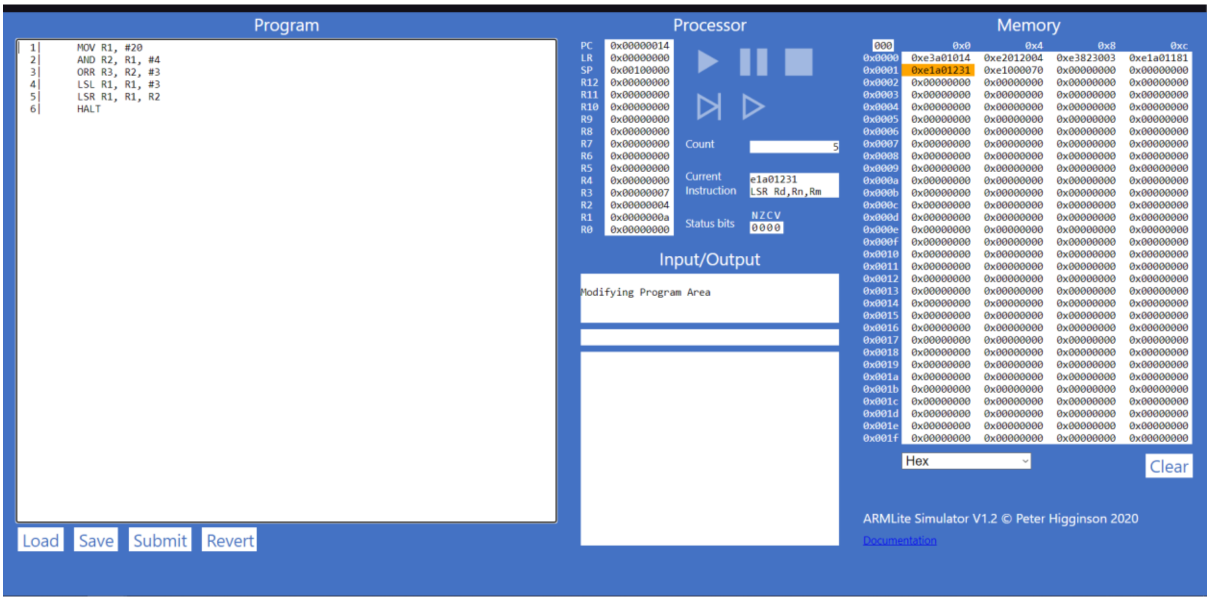
```
MOV R0,#21
SUB R1,R0,#18
SUB R2, R1, #5
ADD R3, R2, R2
ADD R4, R3, R2
ADD R7, R4, #300
HALT
```

PC	0x0000001c
LR	0x00000000
SP	0x00100000
R12	0x00000000
R11	0x00000000
R10	0x00000000
R9	0x00000000
R8	0x00000000
R7	0x00000126
R6	0x00000000
R5	0x00000000
R4	0xfffffffa
R3	0xfffffffc
R2	0xfffffffe
R1	0x00000003
R0	0x00000015

```
Program HALTED. STOP, LOAD or EDIT
```

000	0x0	0x4	0x8	0xc
0x0000	0xe3a00015	0xe2401012	0xe2412005	0xe0823002
0x0001	0xe0834002	0xe2847f4b	0xe1000070	0x00000000
0x0002	0x00000000	0x00000000	0x00000000	0x00000000
0x0003	0x00000000	0x00000000	0x00000000	0x00000000
0x0004	0x00000000	0x00000000	0x00000000	0x00000000
0x0005	0x00000000	0x00000000	0x00000000	0x00000000
0x0006	0x00000000	0x00000000	0x00000000	0x00000000
0x0007	0x00000000	0x00000000	0x00000000	0x00000000
0x0008	0x00000000	0x00000000	0x00000000	0x00000000
0x0009	0x00000000	0x00000000	0x00000000	0x00000000
0x000a	0x00000000	0x00000000	0x00000000	0x00000000
0x000b	0x00000000	0x00000000	0x00000000	0x00000000
0x000c	0x00000000	0x00000000	0x00000000	0x00000000
0x000d	0x00000000	0x00000000	0x00000000	0x00000000
0x000e	0x00000000	0x00000000	0x00000000	0x00000000
0x000f	0x00000000	0x00000000	0x00000000	0x00000000
0x0010	0x00000000	0x00000000	0x00000000	0x00000000
0x0011	0x00000000	0x00000000	0x00000000	0x00000000
0x0012	0x00000000	0x00000000	0x00000000	0x00000000
0x0013	0x00000000	0x00000000	0x00000000	0x00000000
0x0014	0x00000000	0x00000000	0x00000000	0x00000000
0x0015	0x00000000	0x00000000	0x00000000	0x00000000
0x0016	0x00000000	0x00000000	0x00000000	0x00000000
0x0017	0x00000000	0x00000000	0x00000000	0x00000000
0x0018	0x00000000	0x00000000	0x00000000	0x00000000
0x0019	0x00000000	0x00000000	0x00000000	0x00000000
0x001a	0x00000000	0x00000000	0x00000000	0x00000000
0x001b	0x00000000	0x00000000	0x00000000	0x00000000
0x001c	0x00000000	0x00000000	0x00000000	0x00000000
0x001d	0x00000000	0x00000000	0x00000000	0x00000000
0x001e	0x00000000	0x00000000	0x00000000	0x00000000
0x001f	0x00000000	0x00000000	0x00000000	0x00000000

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.***



```
MOV R1, #20
AND R2, R1, #4
ORR R3, R2, #3
LSL R1, R1, #3
LSR R1, R1, R2
HALT
```

```
PC   0x00000014
LR   0x00000000
SP   0x00100000
R12  0x00000000
R11  0x00000000
R10  0x00000000
R9   0x00000000
R8   0x00000000
R7   0x00000000
R6   0x00000000
R5   0x00000000
R4   0x00000000
R3   0x00000007
R2   0x00000004
R1   0x0000000a
R0   0x00000000
```

Count 5

Current Instruction e1a01231 LSR Rd,Rn,Rm

Status bits NZCV 0000

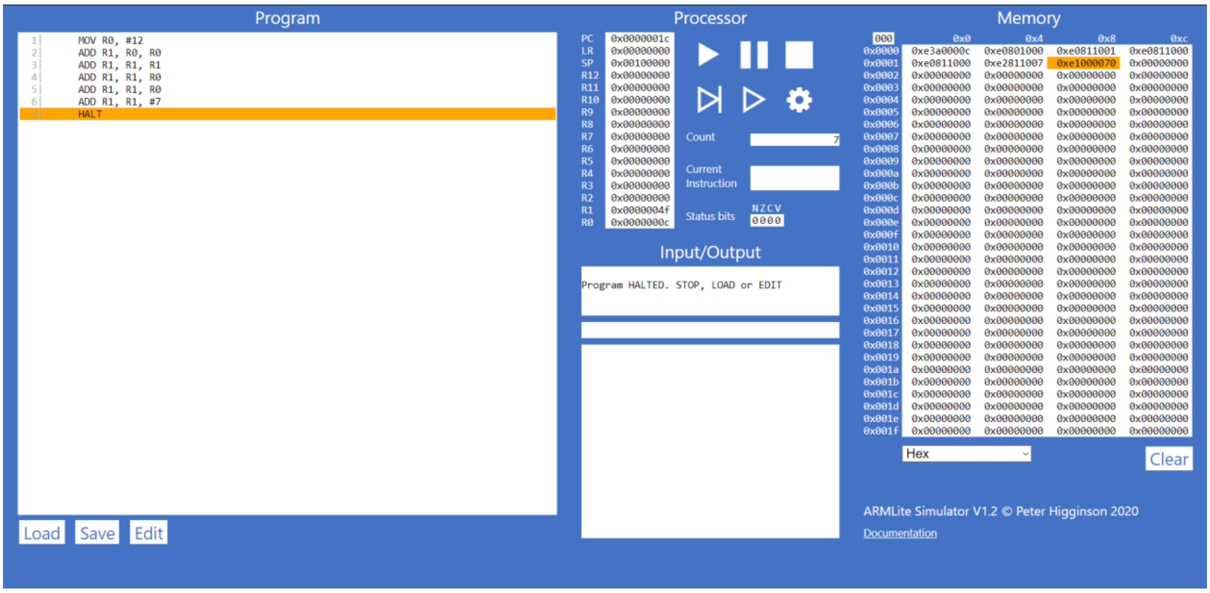
Modifying Program Area

```
000           0x0         0x4         0x8         0xc
0x0000  0xe3a01014  0xe2012004  0xe3823003  0xe1a01181
0x0001  0xe1a01231  0xe1000070  0x00000000  0x00000000
0x0002  0x00000000  0x00000000  0x00000000  0x00000000
0x0003  0x00000000  0x00000000  0x00000000  0x00000000
0x0004  0x00000000  0x00000000  0x00000000  0x00000000
0x0005  0x00000000  0x00000000  0x00000000  0x00000000
0x0006  0x00000000  0x00000000  0x00000000  0x00000000
0x0007  0x00000000  0x00000000  0x00000000  0x00000000
0x0008  0x00000000  0x00000000  0x00000000  0x00000000
0x0009  0x00000000  0x00000000  0x00000000  0x00000000
0x000a  0x00000000  0x00000000  0x00000000  0x00000000
0x000b  0x00000000  0x00000000  0x00000000  0x00000000
0x000c  0x00000000  0x00000000  0x00000000  0x00000000
0x000d  0x00000000  0x00000000  0x00000000  0x00000000
0x000e  0x00000000  0x00000000  0x00000000  0x00000000
0x000f  0x00000000  0x00000000  0x00000000  0x00000000
0x0010  0x00000000  0x00000000  0x00000000  0x00000000
0x0011  0x00000000  0x00000000  0x00000000  0x00000000
0x0012  0x00000000  0x00000000  0x00000000  0x00000000
0x0013  0x00000000  0x00000000  0x00000000  0x00000000
0x0014  0x00000000  0x00000000  0x00000000  0x00000000
0x0015  0x00000000  0x00000000  0x00000000  0x00000000
0x0016  0x00000000  0x00000000  0x00000000  0x00000000
0x0017  0x00000000  0x00000000  0x00000000  0x00000000
0x0018  0x00000000  0x00000000  0x00000000  0x00000000
0x0019  0x00000000  0x00000000  0x00000000  0x00000000
0x001a  0x00000000  0x00000000  0x00000000  0x00000000
0x001b  0x00000000  0x00000000  0x00000000  0x00000000
0x001c  0x00000000  0x00000000  0x00000000  0x00000000
0x001d  0x00000000  0x00000000  0x00000000  0x00000000
0x001e  0x00000000  0x00000000  0x00000000  0x00000000
0x001f  0x00000000  0x00000000  0x00000000  0x00000000
```

Instruction	**Decimal value of the destination register after executing this instruction**	**Binary value of the destination register after executing this instruction**
MOV R1, #20	20	0b00001010
AND R2, R1, #4	4	0b00000100
ORR R3, R2, #3	7	0b00001110
EOR R1, R1, #15	27	0b00011011
LSL R1, R1, #3	216	0b11011000
LSR R1, R1, R2	13	0b00001101

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



```
MOV R0, #12
ADD R1, R0, R0
ADD R1, R1, R1
ADD R1, R1, R0
ADD R1, R1, R0
ADD R1, R1, #7
HALT
```

```
PC   0x0000001c
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   0x0000004f
R0   0x0000000c

Count 7
Status bits NZCV 0000
```

```
Program HALTED. STOP, LOAD or EDIT
```

```
000            0x0         0x4         0x8         0xc
0x0000  0xe3a0000c  0xe0801000  0xe0811001  0xe0811000
0x0001  0xe0811000  0xe2811007  0xe1000070  0x00000000
0x0002  0x00000000  0x00000000  0x00000000  0x00000000
0x0003  0x00000000  0x00000000  0x00000000  0x00000000
0x0004  0x00000000  0x00000000  0x00000000  0x00000000
0x0005  0x00000000  0x00000000  0x00000000  0x00000000
0x0006  0x00000000  0x00000000  0x00000000  0x00000000
0x0007  0x00000000  0x00000000  0x00000000  0x00000000
0x0008  0x00000000  0x00000000  0x00000000  0x00000000
0x0009  0x00000000  0x00000000  0x00000000  0x00000000
0x000a  0x00000000  0x00000000  0x00000000  0x00000000
0x000b  0x00000000  0x00000000  0x00000000  0x00000000
0x000c  0x00000000  0x00000000  0x00000000  0x00000000
0x000d  0x00000000  0x00000000  0x00000000  0x00000000
0x000e  0x00000000  0x00000000  0x00000000  0x00000000
0x000f  0x00000000  0x00000000  0x00000000  0x00000000
0x0010  0x00000000  0x00000000  0x00000000  0x00000000
0x0011  0x00000000  0x00000000  0x00000000  0x00000000
0x0012  0x00000000  0x00000000  0x00000000  0x00000000
0x0013  0x00000000  0x00000000  0x00000000  0x00000000
0x0014  0x00000000  0x00000000  0x00000000  0x00000000
0x0015  0x00000000  0x00000000  0x00000000  0x00000000
0x0016  0x00000000  0x00000000  0x00000000  0x00000000
0x0017  0x00000000  0x00000000  0x00000000  0x00000000
0x0018  0x00000000  0x00000000  0x00000000  0x00000000
0x0019  0x00000000  0x00000000  0x00000000  0x00000000
0x001a  0x00000000  0x00000000  0x00000000  0x00000000
0x001b  0x00000000  0x00000000  0x00000000  0x00000000
0x001c  0x00000000  0x00000000  0x00000000  0x00000000
0x001d  0x00000000  0x00000000  0x00000000  0x00000000
0x001e  0x00000000  0x00000000  0x00000000  0x00000000
0x001f  0x00000000  0x00000000  0x00000000  0x00000000
```

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

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

It show negative decimal number because we shifted the value in R0 by #18 which flip the value of 9999 backward and gave us this value in binary `0b10011100001111000000000000000000`, however ARMlite interpret number in Little Endian thus this number is a negative number and display the value of `-1673789440`.

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.

0b00000001 : 1

0b11111111: -1

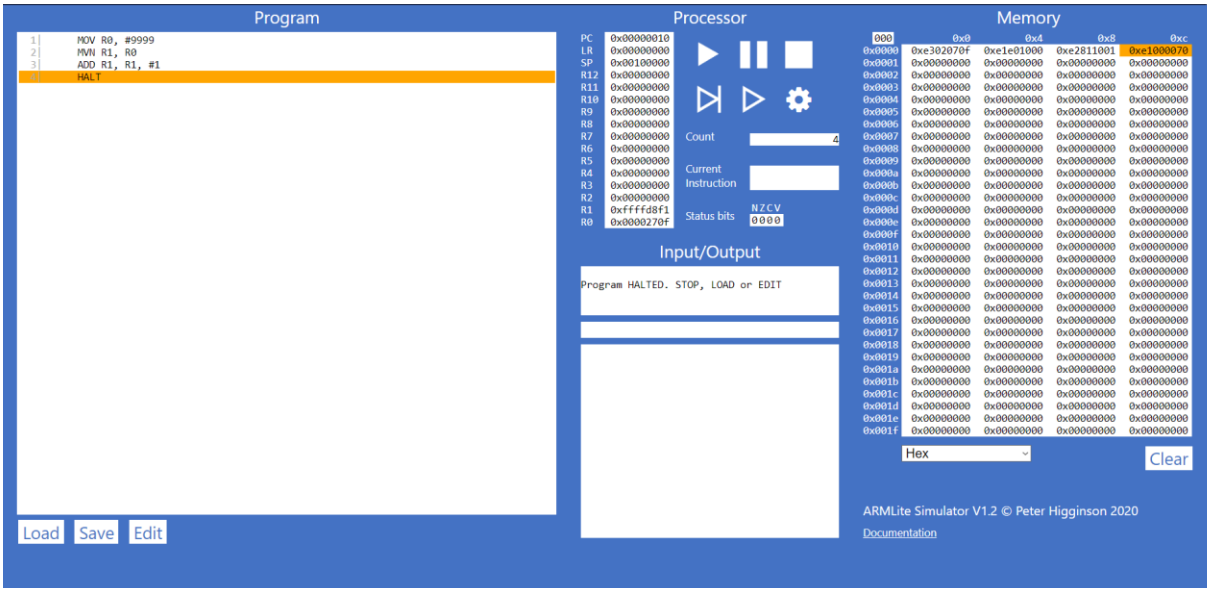
0b00000010: 2

0b11111101: -2

The relation between the two are profound, as the negative decimal value of its positive value are just flip version of the positive counterpart with the part that represent the positive decimal value remains unflip.

It does represent 2’s Compliment for signed integer values.

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.



```
      MOV R0, #9999
      MVN R1, R0
      ADD R1, R1, #1
      HALT
```

```
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   0xffffd8f1
R0   0x0000270f

Status bits  N Z C V
             0 0 0 0
```

```
Program HALTED. STOP, LOAD or EDIT
```

```
000            0x0         0x4         0x8         0xc
0x0000  0xe302070f  0xe1e01000  0xe2811001  0xe1000070
0x0001  0x00000000  0x00000000  0x00000000  0x00000000
0x0002  0x00000000  0x00000000  0x00000000  0x00000000
0x0003  0x00000000  0x00000000  0x00000000  0x00000000
0x0004  0x00000000  0x00000000  0x00000000  0x00000000
0x0005  0x00000000  0x00000000  0x00000000  0x00000000
0x0006  0x00000000  0x00000000  0x00000000  0x00000000
0x0007  0x00000000  0x00000000  0x00000000  0x00000000
0x0008  0x00000000  0x00000000  0x00000000  0x00000000
0x0009  0x00000000  0x00000000  0x00000000  0x00000000
0x000a  0x00000000  0x00000000  0x00000000  0x00000000
0x000b  0x00000000  0x00000000  0x00000000  0x00000000
0x000c  0x00000000  0x00000000  0x00000000  0x00000000
0x000d  0x00000000  0x00000000  0x00000000  0x00000000
0x000e  0x00000000  0x00000000  0x00000000  0x00000000
0x000f  0x00000000  0x00000000  0x00000000  0x00000000
0x0010  0x00000000  0x00000000  0x00000000  0x00000000
0x0011  0x00000000  0x00000000  0x00000000  0x00000000
0x0012  0x00000000  0x00000000  0x00000000  0x00000000
0x0013  0x00000000  0x00000000  0x00000000  0x00000000
0x0014  0x00000000  0x00000000  0x00000000  0x00000000
0x0015  0x00000000  0x00000000  0x00000000  0x00000000
0x0016  0x00000000  0x00000000  0x00000000  0x00000000
0x0017  0x00000000  0x00000000  0x00000000  0x00000000
0x0018  0x00000000  0x00000000  0x00000000  0x00000000
0x0019  0x00000000  0x00000000  0x00000000  0x00000000
0x001a  0x00000000  0x00000000  0x00000000  0x00000000
0x001b  0x00000000  0x00000000  0x00000000  0x00000000
0x001c  0x00000000  0x00000000  0x00000000  0x00000000
0x001d  0x00000000  0x00000000  0x00000000  0x00000000
0x001e  0x00000000  0x00000000  0x00000000  0x00000000
0x001f  0x00000000  0x00000000  0x00000000  0x00000000
```

ARMLite Simulator V1.2 © Peter Higginson 2020

Works just like 2’s Compliment.