Name : Nguyen Quoc Thang

ID : 104193360

LAB 7 : Submission

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

7.1.1 What value is displayed ? Why ?

The memory display 0x00000065 because it convert 101 decemal into hexadecimal.

Click on another memory word, enter 0x101

7.1.2 What value is displayed, and why?

Lalue starting with 0x means the value is represented in hexadecimal so the word memory will display 0x00000101

On another memory word, enter 0b101

7.1.3 What value is displayed, and why?

The memory display 0x00000005 because it convert 0b101 form binary to hexadecimal.

The Tooltip function display binary number of hexadecimal memory of value that have been filled in memory before

When changee the base from hexadecimal from Decimal(unsigned) the Tooltip function display both

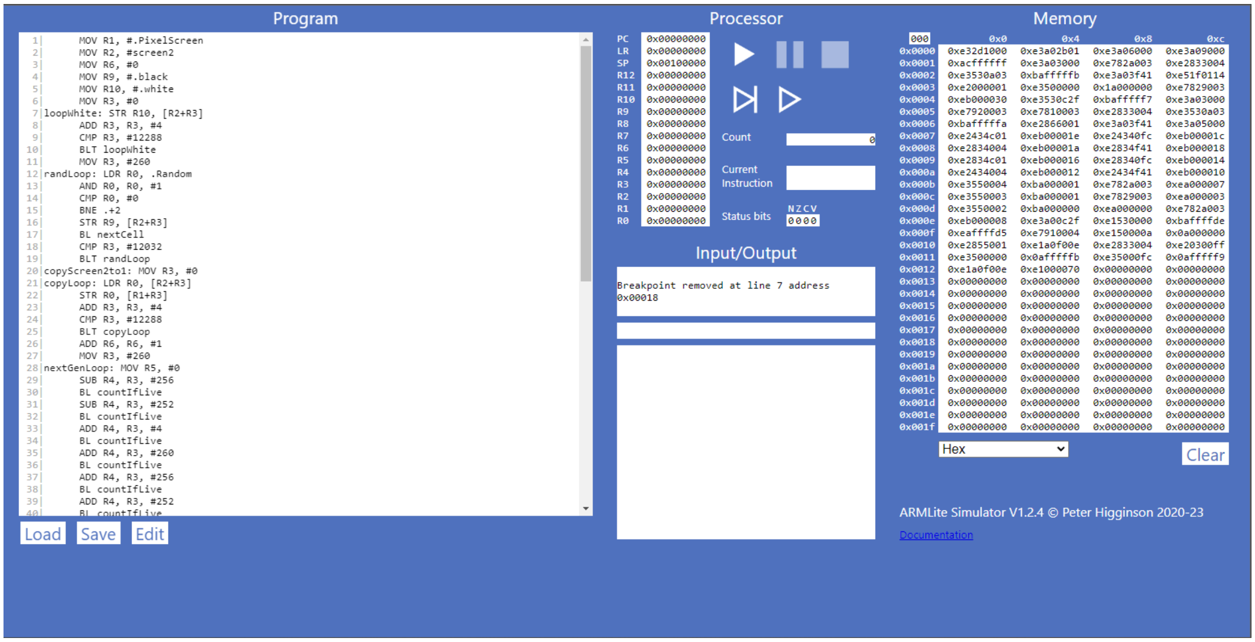
7.1.4: Does changing the representation of the data in memory also change the representation of the row and column-headers (the white digits on a blue background)? Should it ?

Changing the representation of the data in memory does not changes the representation of the row and column because row and column headers usually represent the address of the memory, which is independent of the memory's value

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

Each memory word has 32 bits it is equivalent to 4 bytes so this column header memory address offsets increases by the number set of 0x4

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



```
 1|      MOV R1, #.PixelScreen
 2|      MOV R2, #screen2
 3|      MOV R6, #0
 4|      MOV R9, #.black
 5|      MOV R10, #.white
 6|      MOV R3, #0
 7|loopWhite: STR R10, [R2+R3]
 8|      ADD R3, R3, #4
 9|      CMP R3, #12288
10|      BLT loopWhite
11|      MOV R3, #260
12|randLoop: LDR R0, .Random
13|      AND R0, R0, #1
14|      CMP R0, #0
15|      BNE .+2
16|      STR R9, [R2+R3]
17|      BL nextCell
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 R6, R6, #1
27|      MOV R3, #260
28|nextGenLoop: MOV R5, #0
29|      SUB R4, R3, #256
30|      BL countIfLive
31|      SUB R4, R3, #252
32|      BL countIfLive
33|      ADD R4, R3, #4
34|      BL countIfLive
35|      ADD R4, R3, #260
36|      BL countIfLive
37|      ADD R4, R3, #256
38|      BL countIfLive
39|      ADD R4, R3, #252
40|      BL countIfLive
```

```
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
```

```
Breakpoint removed at line 7 address
0x00018
```

```
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
```

ARMLite Simulator V1.2.4 © Peter Higginson 2020-23

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

After the code has been submitted, the assembler proram convert the source code into machine code and then the machine code was stored into the memory space

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 hexadecimal values appears whenever I hover the mouse over one of the line numbers of the source code. It represents the location of the memory space that contains the instruction.

<u>The answer to the try inserting part is below question 7.3.3</u>

-The blank lines disappeared and it does not save in the memory space.

- When we add comment on a line of its own or after an instruction after the code has sumitted the comment will change the color difference from the code

- If add the comment on a line , the line numbers increases with the number of comments added

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

The highlighting indicates what line of code run in the program and the highlighting in the memory represents for the location where the line of code is stored in the memory space.

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

When I click the button circle red , the instructions run step by step , line by line.

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

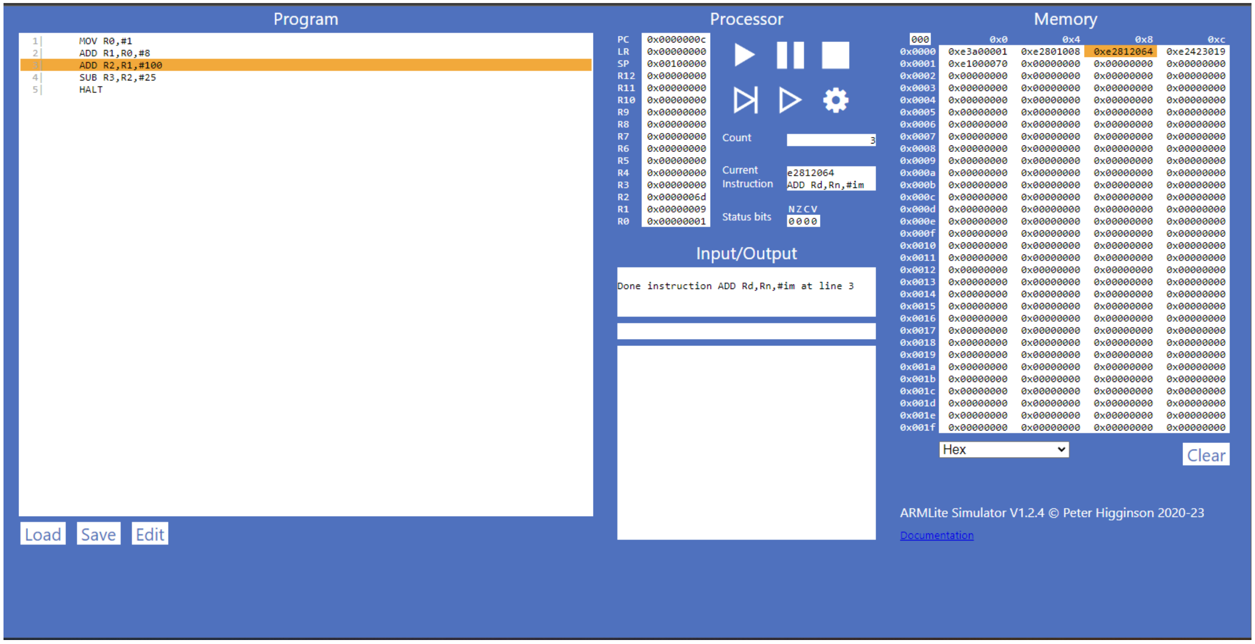
The processor pause jusst before executing the line with the breakpoint .

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  The value 1 is moved to register 0 .  R0 will be equal to 1

ADD R1,R0,#8  The value R0 is added to 8 and the result will be stored in R1

ADD R2,R1,#100 The value R0 is added to 100 and the result wil be store in R2

SUB R3,R2,#25 The value of R2 subtracted fron 25, the result will be store in R3, the
value of R3 in hex will display 54 but in decimal it will display 84
```

-Value of R0 will be 1 and value of R1 will be 9 when it is complete.

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



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

PC	0x0000000c
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	0x0000006d
R1	0x00000009
R0	0x00000001

Count: 3

Current Instruction: e2812064 ADD Rd,Rn,#im

Status bits: N Z C V 0 0 0 0

```
Done instruction ADD Rd,Rn,#im at line 3
```

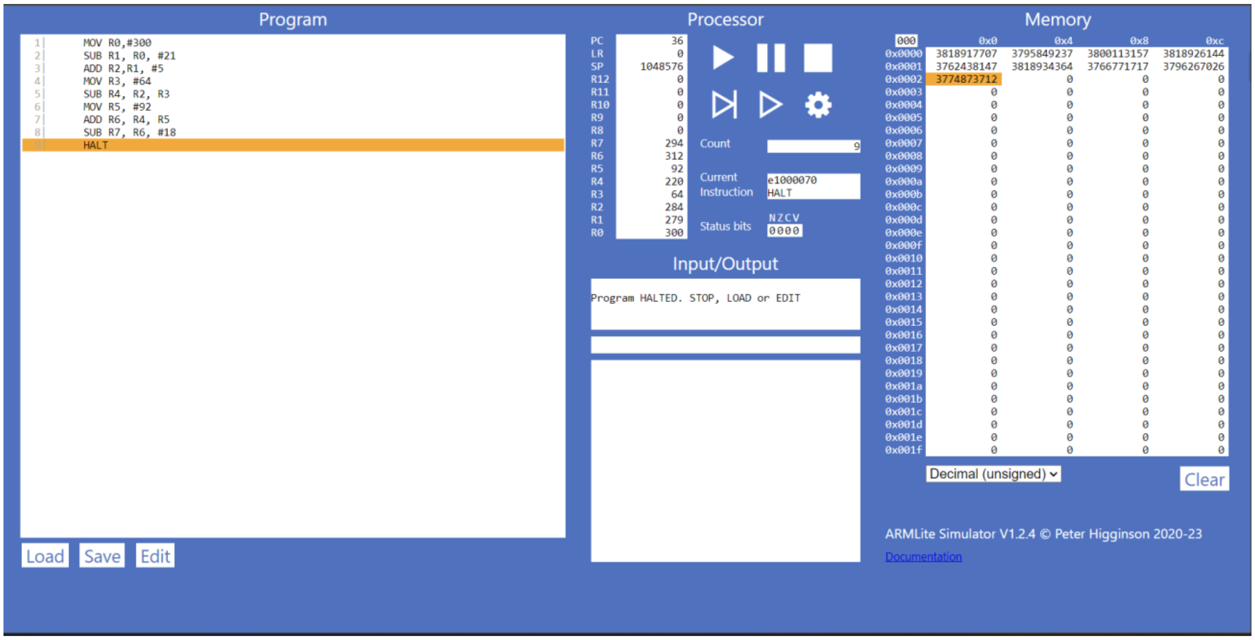
000	0x0	0x4	0x8	0xc
0x0000	0xe3a00001	0xe2801008	0xe2812064	0xe2423019
0x0001	0xe1000070	0x00000000	0x00000000	0x00000000
0x0002	0x00000000	0x00000000	0x00000000	0x00000000
0x0003	0x00000000	0x00000000	0x00000000	0x00000000
0x0004	0x00000000	0x00000000	0x00000000	0x00000000
0x0005	0x00000000	0x00000000	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

Hex

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)*



```
MOV R0,#300
SUB R1, R0, #21
ADD R2,R1, #5
MOV R3, #64
SUB R4, R2, R3
MOV R5, #92
ADD R6, R4, R5
SUB R7, R6, #18
HALT
```

```
PC        36
LR         0
SP   1048576
R12        0
R11        0
R10        0
R9         0
R8         0
R7       294
R6       312
R5        92
R4       220
R3        64
R2       284
R1       279
R0       300

Count                 9
Current Instruction   e1000070
                      HALT
Status bits           NZCV
                      0000
```

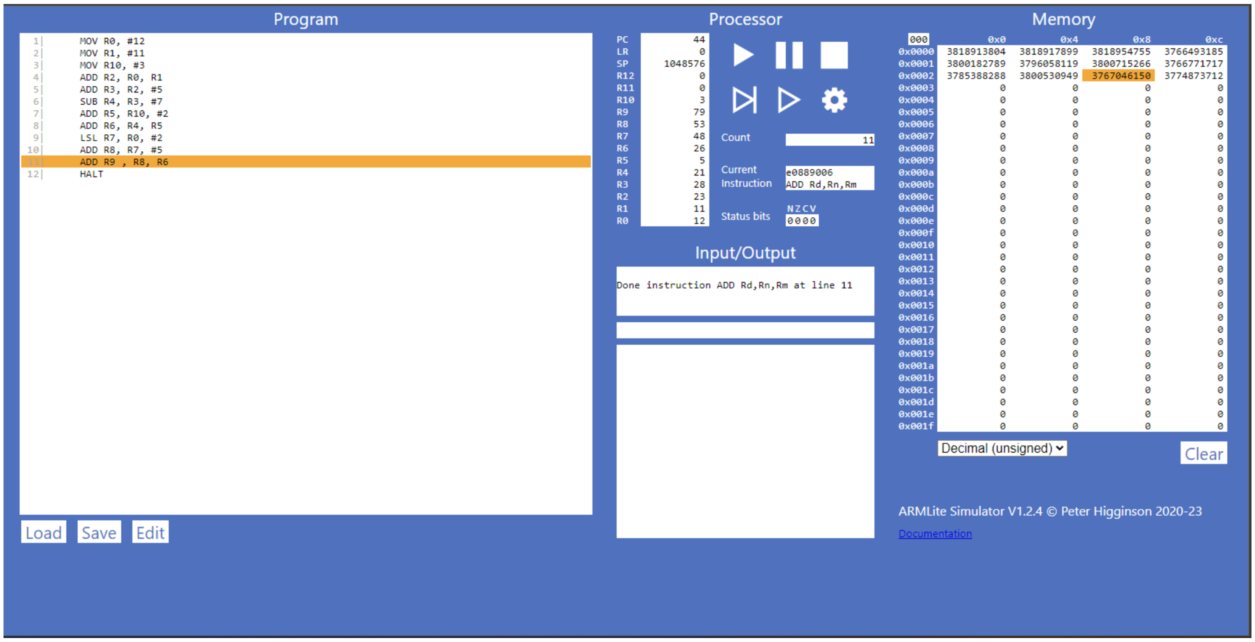
```
Program HALTED. STOP, LOAD or EDIT
```

```
   000          0x0         0x4         0x8         0xc
0x0000   3818917707  3795849237  3800113157  3818926144
0x0001   3762438147  3818934364  3766771717  3796267026
0x0002   3774873712           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
```

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.***

Instruction	**Decimal value of the destination register after executing this instruction**	**Binary value of the destination register after executing this instruction**
MOV R0,#13	13	00001101
AND R1, R0, #10	8	00000100
ORR R2, R1, #3	11	00001011
EOR R3, R1, R0	5	00000101
LSL R4, R1, #3	64	01000000
LSR R5, R0, #5	0	00000000

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).



```
        MOV R0, #12
        MOV R1, #11
        MOV R10, #3
        ADD R2, R0, R1
        ADD R3, R2, #5
        SUB R4, R3, #7
        ADD R5, R10, #2
        ADD R6, R4, R5
        LSL R7, R0, #2
        ADD R8, R7, #5
        ADD R9 , R8, R6
        HALT
```

```
PC         44
LR          0
SP    1048576
R12         0
R11         0
R10         3
R9         79
R8         53
R7         48
R6         26
R5          5
R4         21
R3         28
R2         23
R1         11
R0         12
```

Count 11

Current Instruction e0889006 ADD Rd,Rn,Rm

Status bits NZCV 0000

```
Done instruction ADD Rd,Rn,Rm at line 11
```

```
   000         0x0        0x4        0x8        0xc
0x0000  3818913804 3818917899 3818954755 3766493185
0x0001  3800182789 3796058119 3800715266 3766771717
0x0002  3785388288 3800530949 3767046150 3774873712
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
```

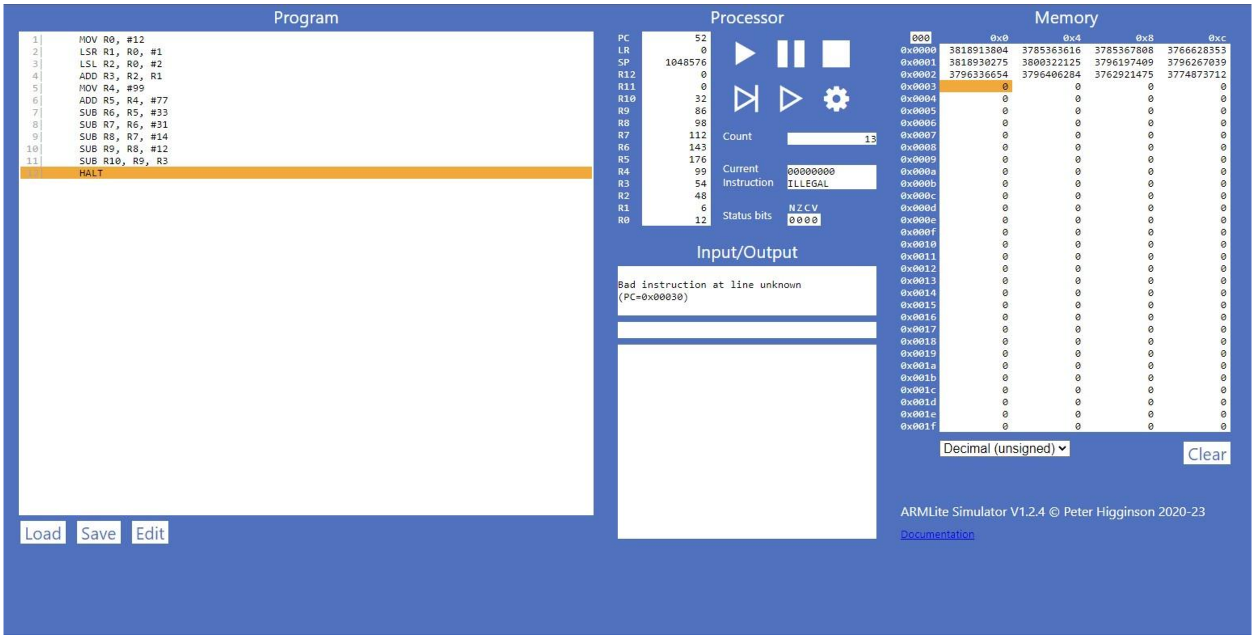
ARMLite Simulator V1.2.4 © Peter Higginson 2020-23

Documentation

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

When the program is complete, take a screen shot of the code and the register table and paste into your submission document.



```
        MOV R0, #12
        LSR R1, R0, #1
        LSL R2, R0, #2
        ADD R3, R2, R1
        MOV R4, #99
        ADD R5, R4, #77
        SUB R6, R5, #33
        SUB R7, R6, #31
        SUB R8, R7, #14
        SUB R9, R8, #12
        SUB R10, R9, R3
        HALT
```

```
PC         52
LR          0
SP    1048576
R12         0
R11         0
R10        32
R9         86
R8         98
R7        112
R6        143
R5        176
R4         99
R3         54
R2         48
R1          6
R0         12
```

Count 13

Current Instruction 00000000 ILLEGAL

Status bits NZCV 0000

```
Bad instruction at line unknown
(PC=0x00030)
```

```
   000         0x0        0x4        0x8        0xc
0x0000  3818913804 3785363616 3785367808 3766628353
0x0001  3818930275 3800322125 3796197409 3796267039
0x0002  3796336654 3796406284 3762921475 3774873712
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
```

ARMLite Simulator V1.2.4 © Peter Higginson 2020-23

Documentation

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

The value of R1 is a negative decimal number because the instruction is shifting the 18 bits of the value in R0 to the left. In this case, the value of R0 is 9999, which is equivalent to the binary value 10011100001111. When it is shifted 18 bits to the left, it exceeds the range of positive numbers, resulting in a negative value. This negative value is then stored in R1 because R1 is not directly related to the value 9999.

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.

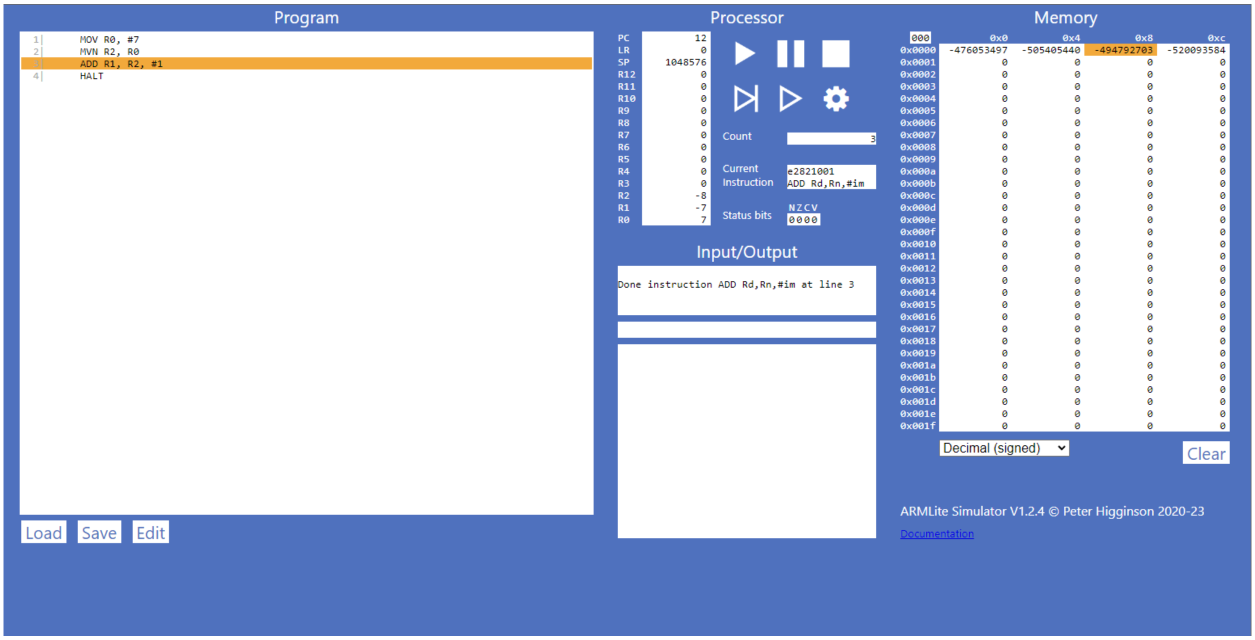
1 ➔ 0b00000000000000000000000000000001

-1 ➔ 0b11111111111111111111111111111111

2 ➔ 0b00000000000000000000000000000010

-2 ➔ 0b11111111111111111111111111111110

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, #7
MVN R2, R0
ADD R1, R2, #1
HALT
```

```
PC        12
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        -8
R1        -7
R0         7

Count                 3
Current Instruction   e2821001
                      ADD Rd,Rn,#im
Status bits           NZCV
                      0000
```

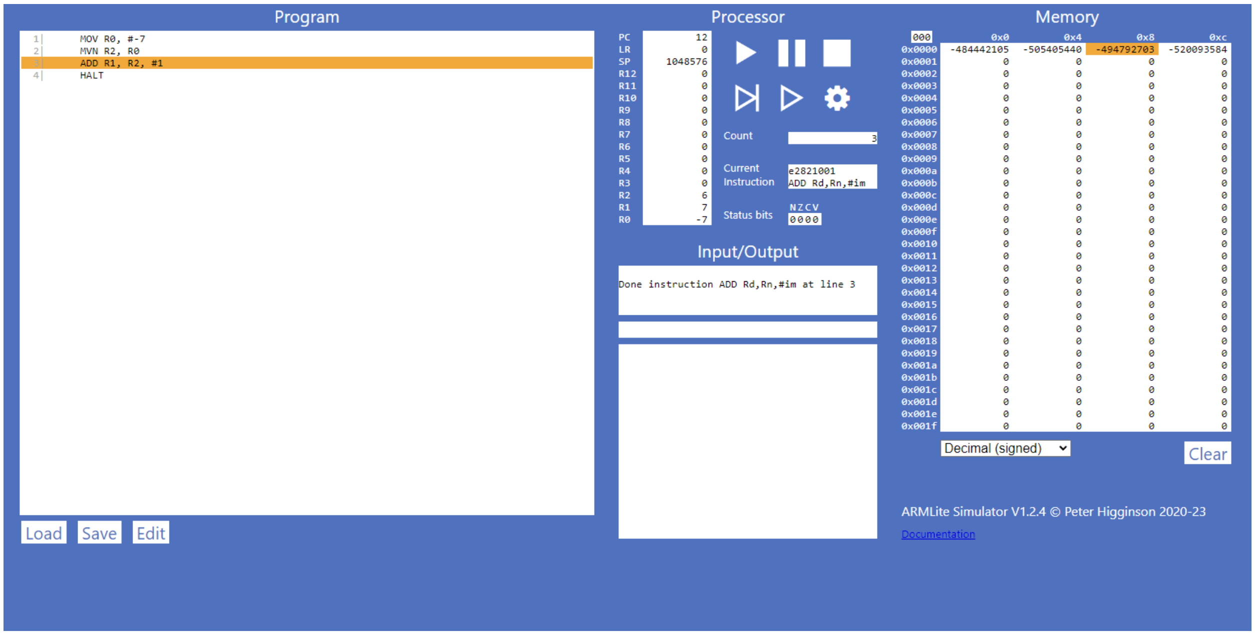
```
Done instruction ADD Rd,Rn,#im at line 3
```

```
000              0x0        0x4        0x8        0xc
0x0000    -476053497 -505405440 -494792703 -520093584
0x0001             0          0          0          0
0x0002             0          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
```

Using the program you wrote above, enter the negative version of the number you previously tested as the input (ie use the output of the previous test as the input).

What do you notice ?

- Negative value has changed to positive value



Program

```
1|      MOV R0, #-7
2|      MVN R2, R0
3|      ADD R1, R2, #1
4|      HALT
```

Processor

PC	12
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	6
R1	7
R0	-7

Count: 3

Current Instruction: e2821001
ADD Rd,Rn,#im

Status bits: NZCV 0000

Input/Output

```
Done instruction ADD Rd,Rn,#im at line 3
```

Memory

000	0x0	0x4	0x8	0xc
0x0000	-484442105	-505405440	-494792703	-520093584
0x0001	0	0	0	0
0x0002	0	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

ARMLite Simulator V1.2.4 © Peter Higginson 2020-23