

7.1.1 What value is displayed? Why?

The value displayed is 0x00000065, because it converts the decimal value to hex.

7.1.2 What value is displayed, and why?

The value displayed is 0x00000101, because this is already in hex, it stays the same.

7.1.3 What value is displayed, and why?

The value displayed is 0x00000005, because it converts the binary into hex.

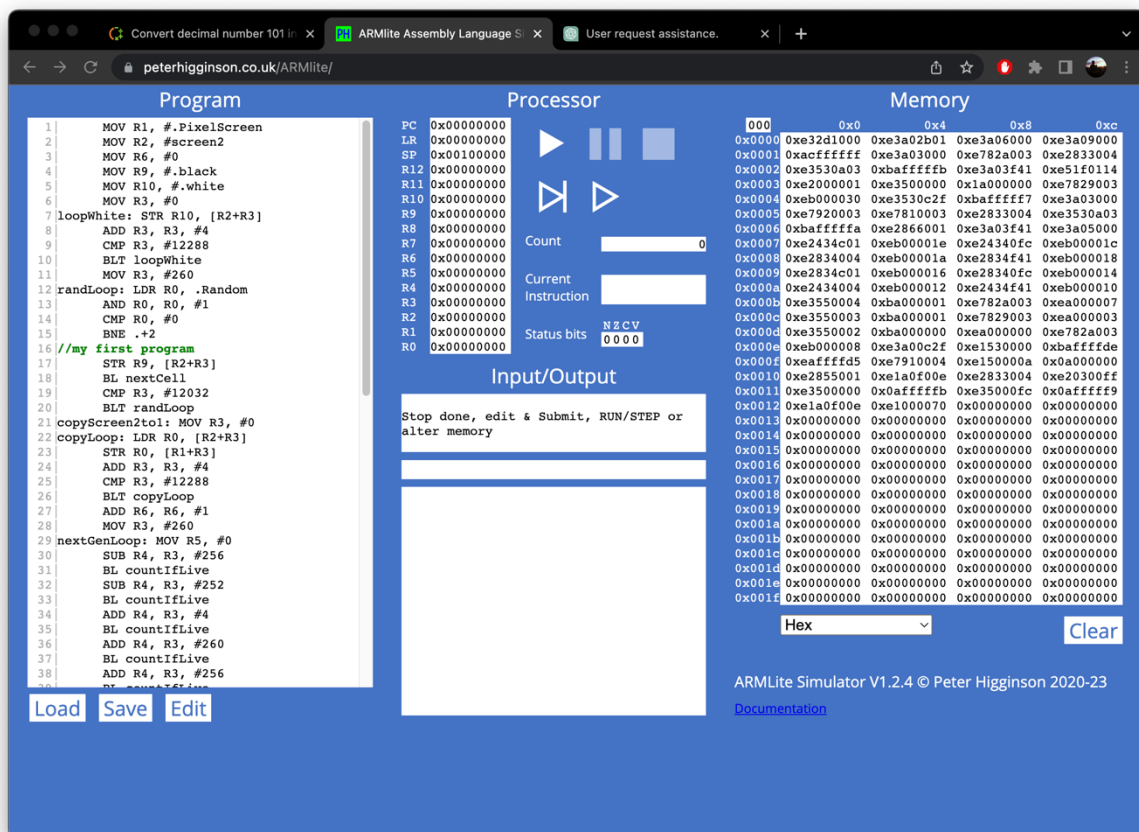
7.1.4: Does changing the representation of the data in memory also change the representation of the row and column-headers (the white digits on a blue background)? Should it ?

No it does not, it only changes how the contents of the memory are displayed and it shouldn't affect the format or representation of the row and column headers.

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

Word-aligned addresses are addresses that are multiples of 4 (0x4) and doubleword-aligned addresses are addresses that are multiples of 8 (0x8). This alignment requirement is due to the way data is fetched and stored in memory by the processor.

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.

When we submit the codes, the simulator would first assemble the code to generate machine code that can be executed by the simulator. The assembly process involves converting each instruction in the source code into its corresponding binary representation. It also resolves symbol names, such as labels and variables, into their corresponding memory addresses.

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 5-digit hex value that appears is likely the memory address of the instruction that corresponds to that line of code.

The blank lines disappear, also the additional spaces.

The comments turn green.

The line numbers incremented by 1 because I inserted a `“//My first program”` into the code.

The program doesn't run.

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

The highlighting signifies where in the code line the simulator is executing.

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

This highlights the next line of code after we press the Pause button.

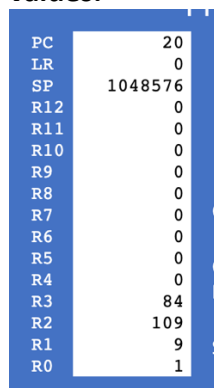
7.4.3 Has the processor paused just before, or just after executing the line with the breakpoint ?

It pauses 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 ?

It's going to execute the content and gives R1 the value 9, because it basically says $R0 + 8$, and the result is stored in R1.

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



PC	20
LR	0
SP	1048576
R12	0
R11	0
R10	0
R9	0
R8	0
R7	0
R6	0
R5	0
R4	0
R3	84
R2	109
R1	9
R0	1

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)

The screenshot shows an assembly simulator with two main panels: 'Program' and 'Processor'.

Program Panel: A list of 8 instructions. Line 7, 'MOV R7, R5', is highlighted in orange.

```

1  MOV R0, #300
2  ADD R1, r0, #92
3  SUB R2, R1, #64
4  SUB R3, R2, #21
5  SUB R4, R3, #18
6  ADD R5, R4, #5
7  MOV R7, R5
8  HALT

```

Processor Panel: Shows the state of registers and other components.

Register	Value
PC	28
LR	0
SP	1048576
R12	0
R11	0
R10	0
R9	0
R8	0
R7	294
R6	0
R5	294
R4	289
R3	307
R2	328
R1	392
R0	300

Other processor information:

- Count: (empty)
- Current Instruction: e1a07005 MOV Rd, Rm
- Status bits: NZCV 0000

Input/Output: (empty)

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, #80	80	0b01010000
MOV R1, #30	30	0b00011110
EOR R0,R0, #35	115	0b01110011
ORR R0,R1,R1	30	0b00011110
LSL R1,R1, #2	120	0b01111000
LSRS R0,R0, #3	3	0b00000011
AND R1,R0, #4	0	0b00000000
HALT		

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).

The screenshot shows an assembly simulator with two main panels: 'Program' and 'Processor'.

Program Panel: A list of 10 instructions. Line 10, 'HALT', is highlighted in orange.

```

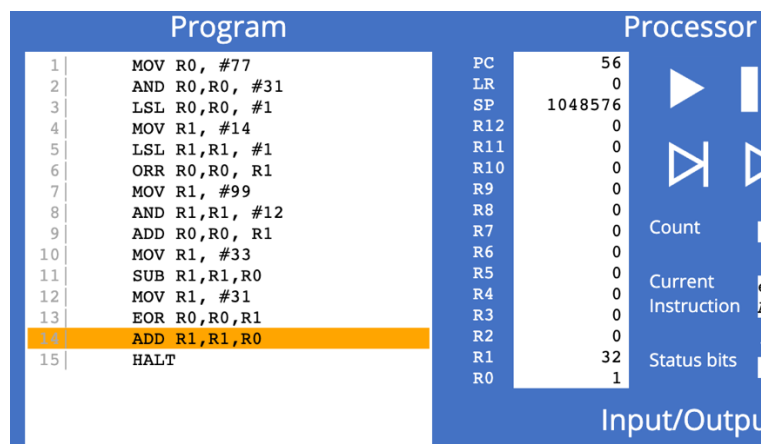
1  MOV R0, #12
2  LSL R0,R0, #2
3  ORR R0,R0, #3
4  EOR R0,R0, #7
5  MOV R1, #2
6  ADD R0, R0, R1
7  AND R0,R0, #11
8  LSL R0,R0, #5
9  ORR R0,R0, #15
10 HALT

```

Processor Panel: Shows the state of registers and other components.

Register	Value
PC	40
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	2
R0	79

Task 7.5.6: Let's play again !



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

Because it is displayed in memory in signed decimal. However, if we interpret the value in R1 as an unsigned integer, which is the default interpretation for logical operations in assembly, then the value will be a large positive number which is (2621177856) we get that from 9999×2^{18} .

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.

Positive number in signed notation is the same as its binary representation in two's complement notation, while the binary representation of a negative number in two's complement notation is obtained by inverting all the bits of its positive counterpart, and then adding 1.

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.

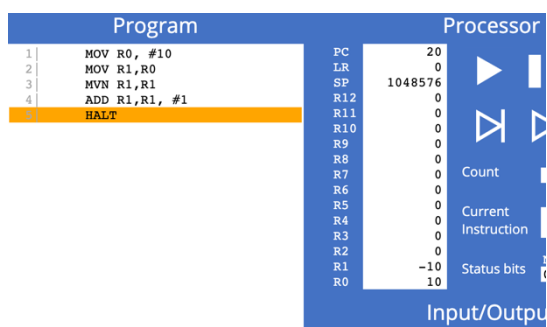


Figure 1: positive to negative.

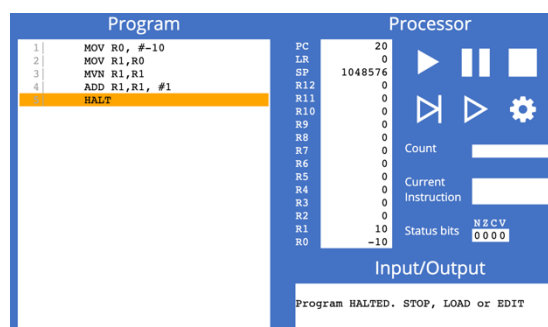


Figure 2: negative to positive