# 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. ecause 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 inputing 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 inputing 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 memoery words in that row.


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


===> 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

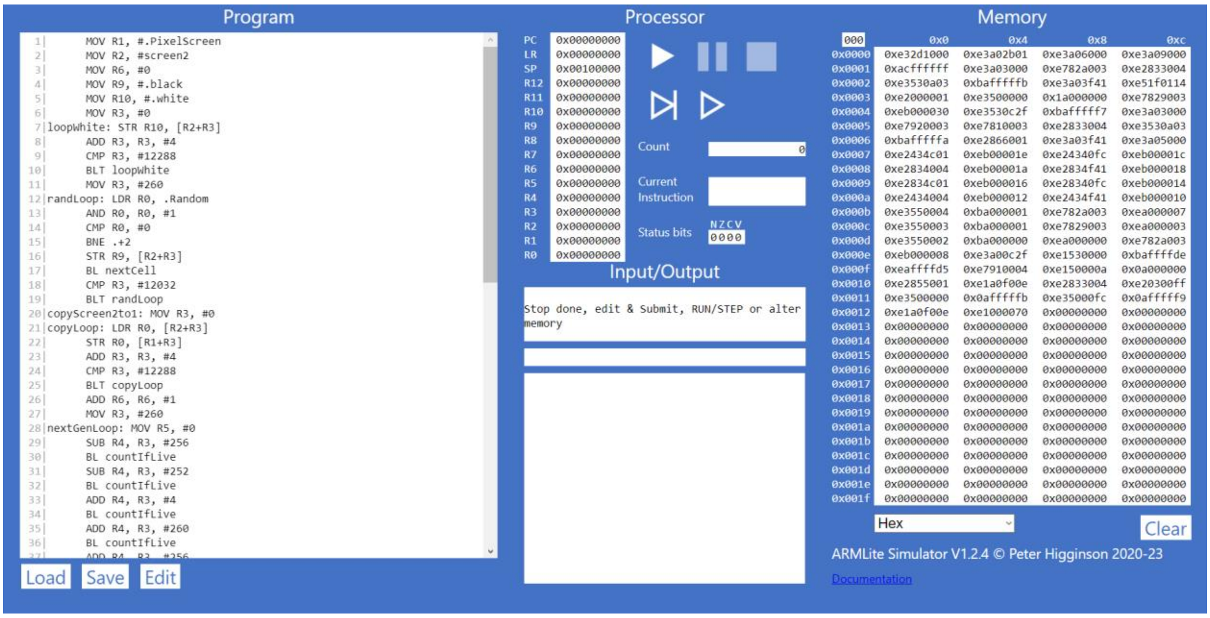
**7.2.1 Notice these column header memory address offsets go up in multiples of 0x4. Why is this ?**

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).

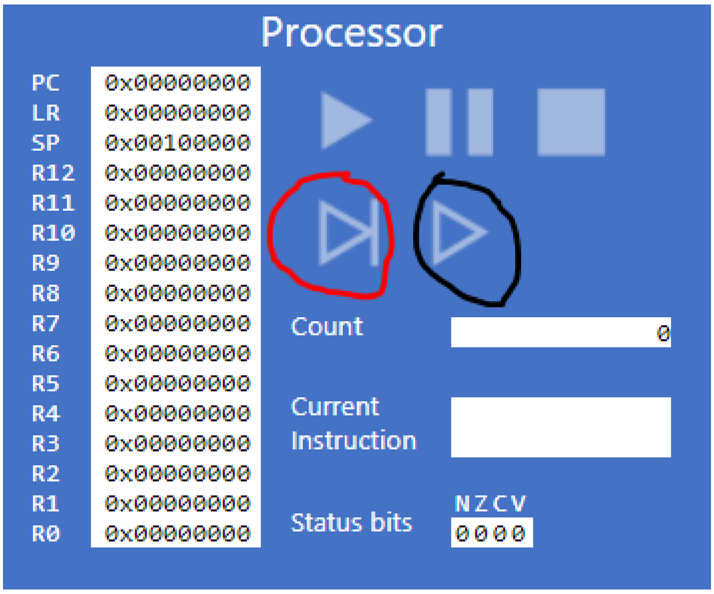
As well as freezing the graphics screen, you will also see orange highlighting appear in both the Program and Memory windows.

### 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.

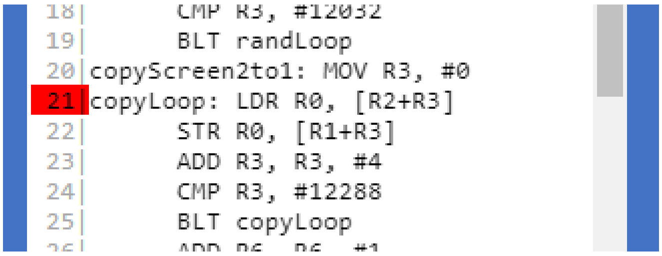



### 7.4.2 What do you think happens when you click the button circled in red ?

- 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:




This is called ‘setting a break point’ 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 ?**

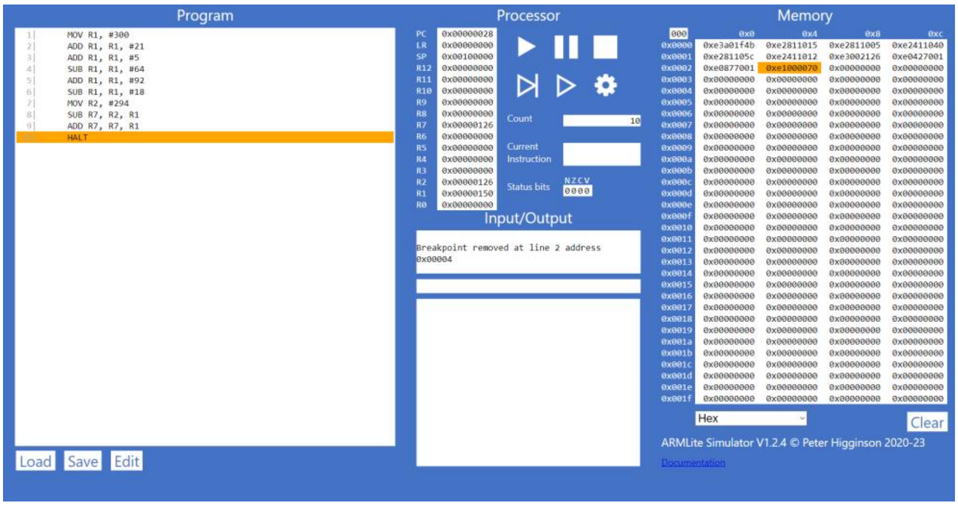
– The processor paused just before executing the line with the breakpoint

# Week 7 Lab - Part 7.5

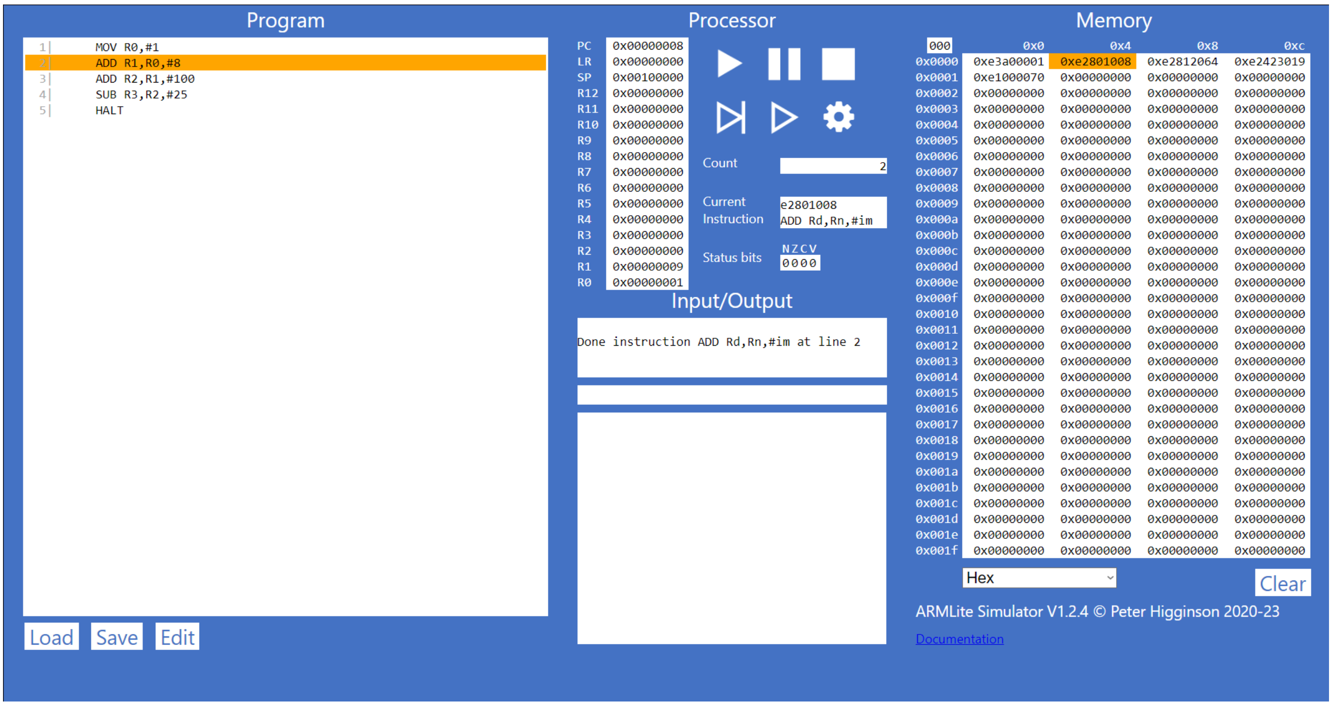
***7.5.1 Before executing this instruction, describe in words what you think this instruction is going to do, and what values you expect to see in R0 and R1 when it is complete ?***

- 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.***




**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)**



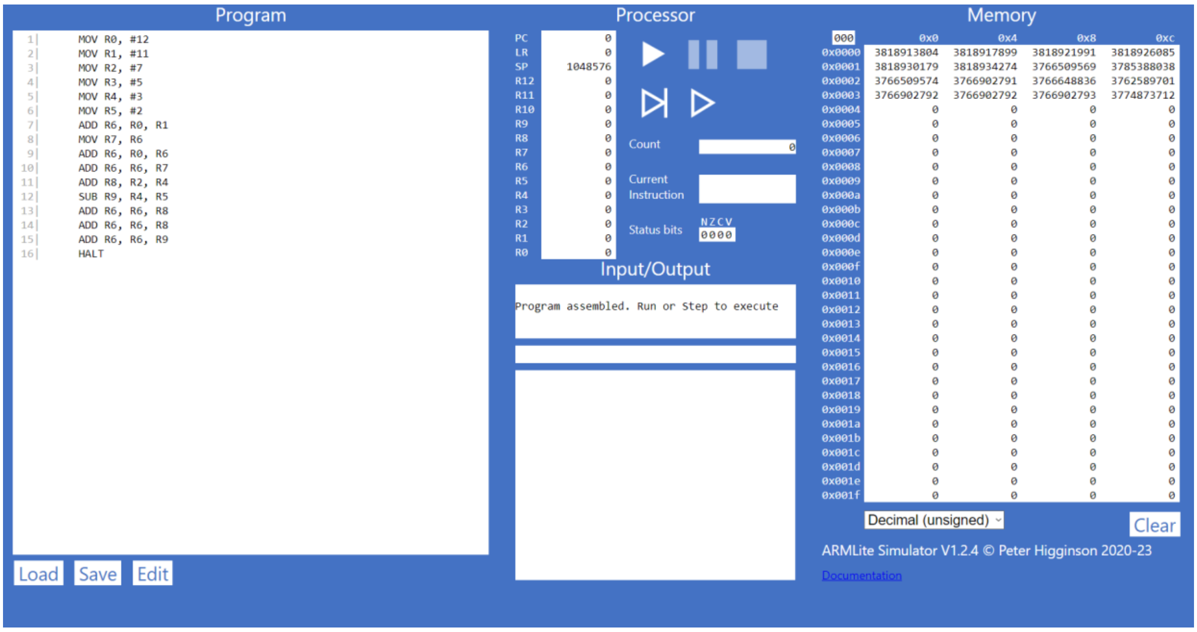
**7.5.4 Task: Write your own simple program, that starts with a MOV (as in the previous example) followed by five instructions, using each of the five new instructions listed above, once only, but in any order you like – plus a HALT at the end, and with whatever immediate values you like.**

**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

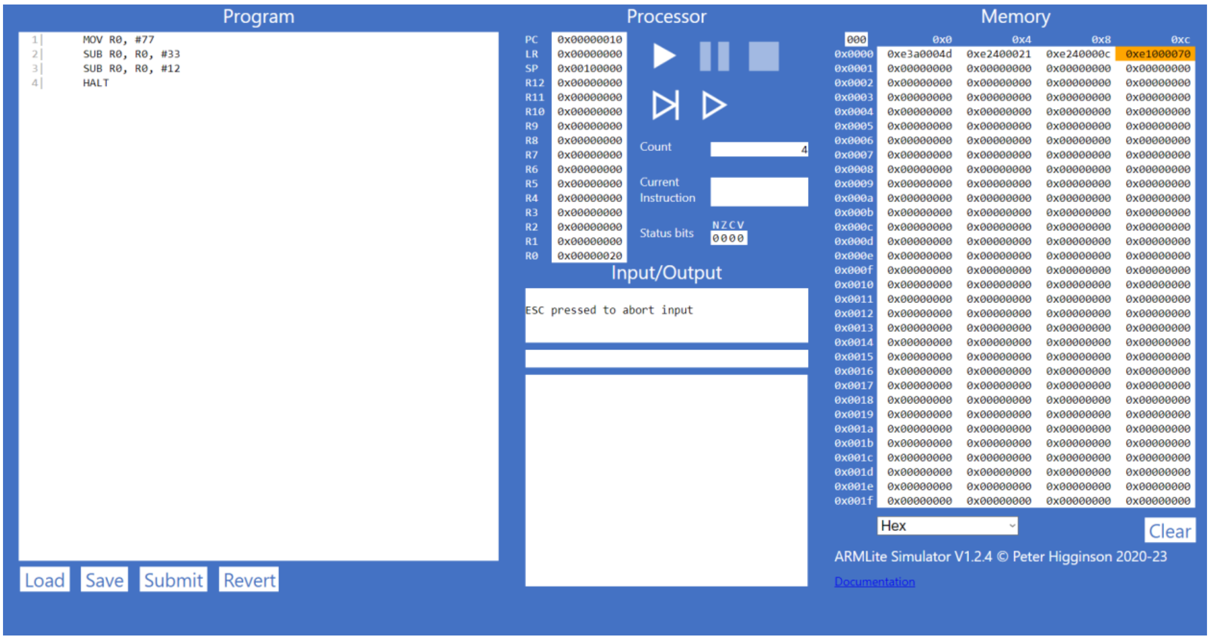



| Instruction | Decimal value of the destination register after executing this instruction | Binary value of the destination register after executing this instruction |
|---|---|---|
| MOV R0, #12 | 12 | 1100 |
| MOV R1, #11 | 11 | 1011 |
| MOV R2, #7 | 7 | 111 |
| MOV R3, #5 | 5 | 101 |
| MOV R4, #3 | 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



| 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.**