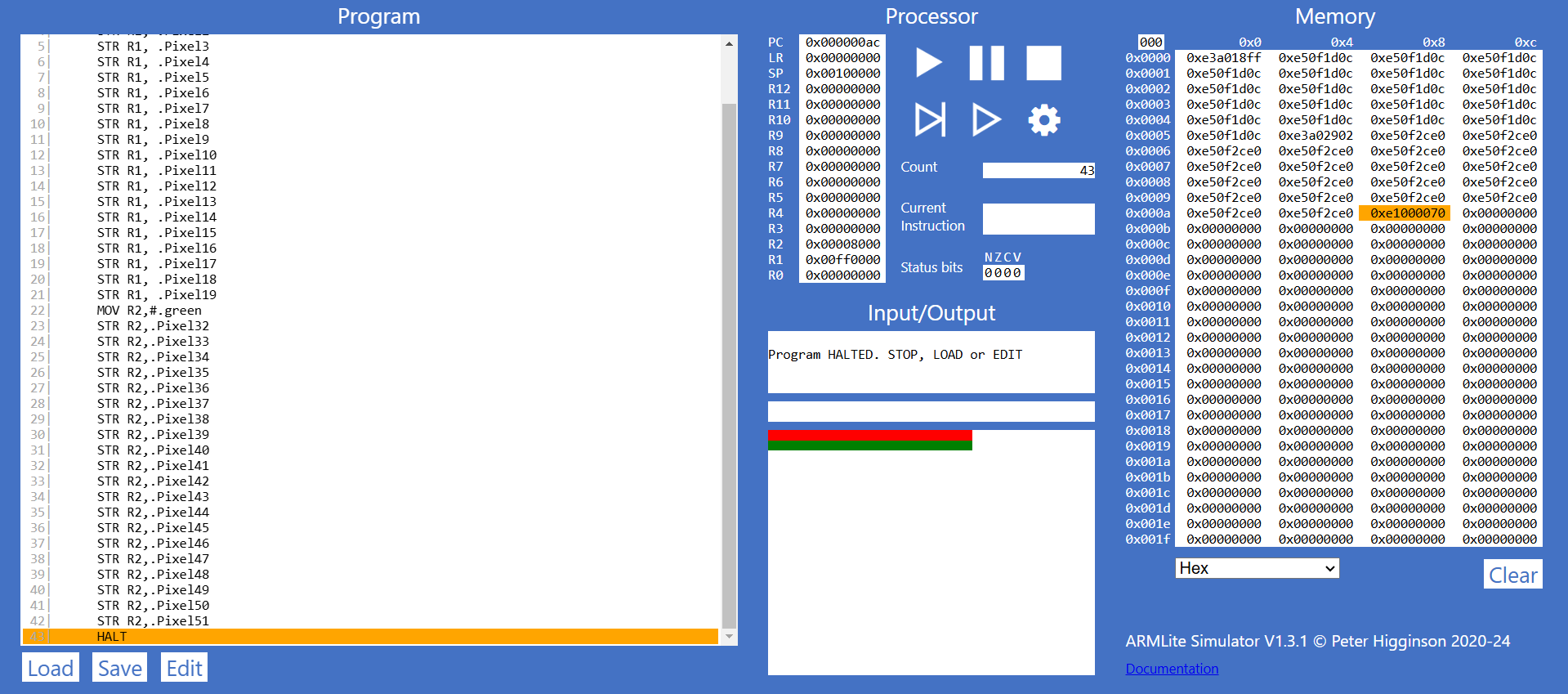
**Faculty of Science, Engineering and Technology**



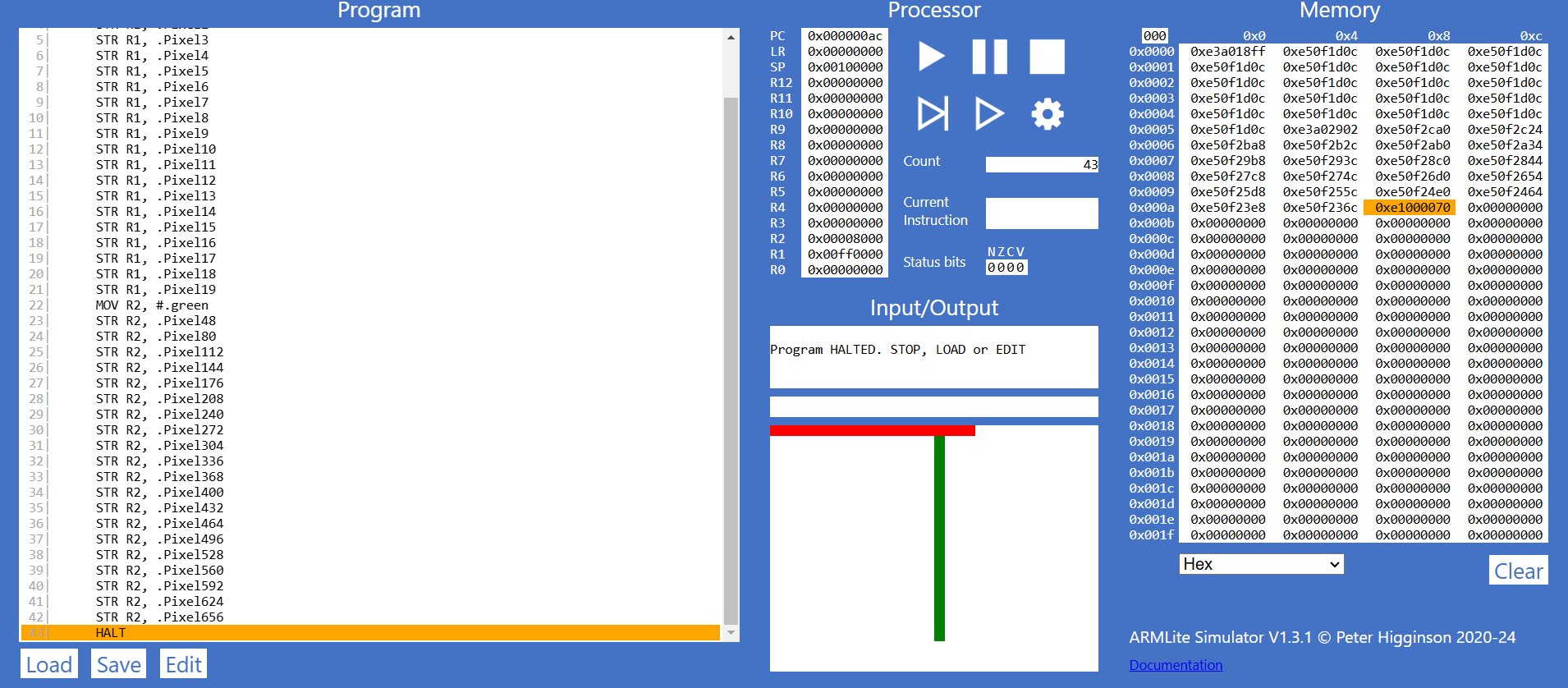
**Computer Systems**

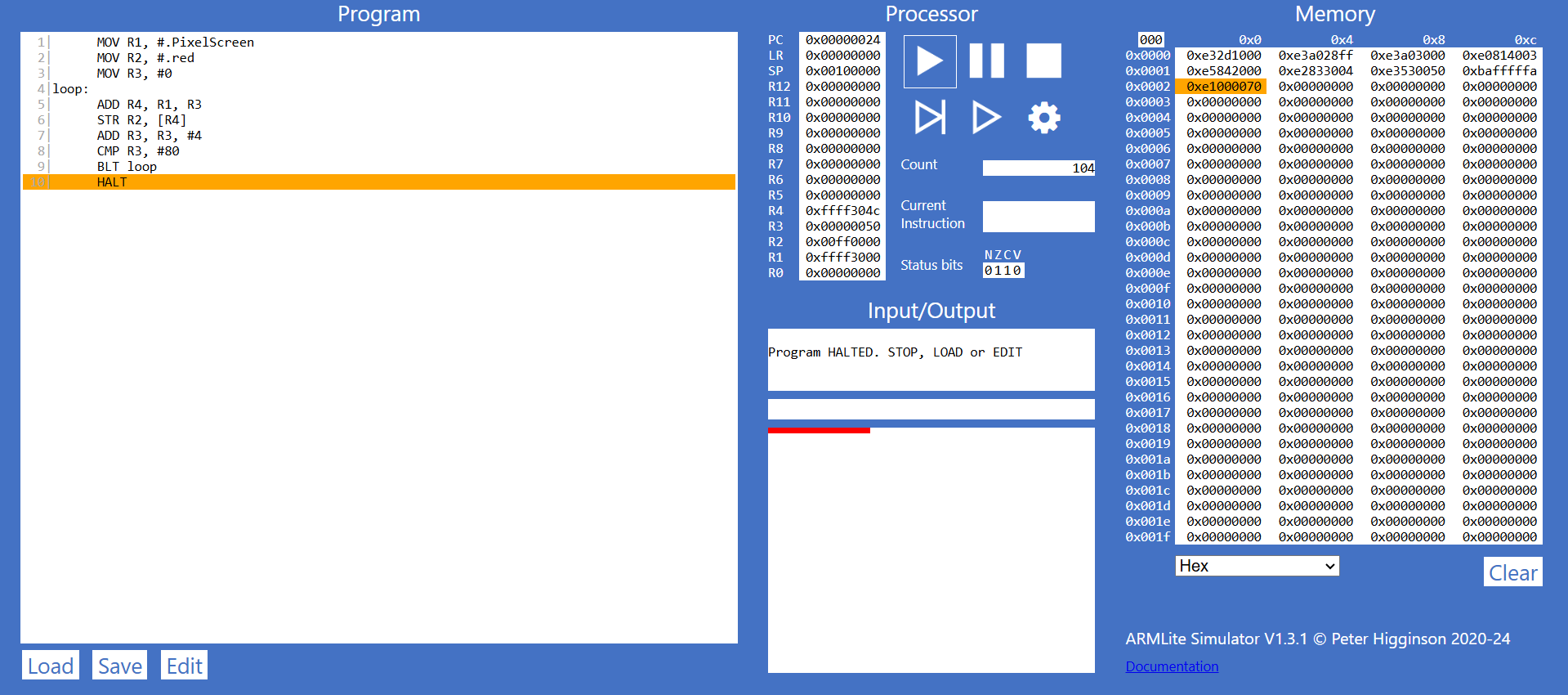
***Week 9***

***Part 9.1  
9.1.1  
(a)***



***(b)***



***9.1.2***

***9.1.3(a)***

==> Explanation of Indirect Addressing:

Indirect addressing is used in this code because the address of the pixel to be drawn is not directly specified in the instruction. Instead, the address is calculated at runtime using the base address stored in a register and an offset. Here’s how it works in the provided code:

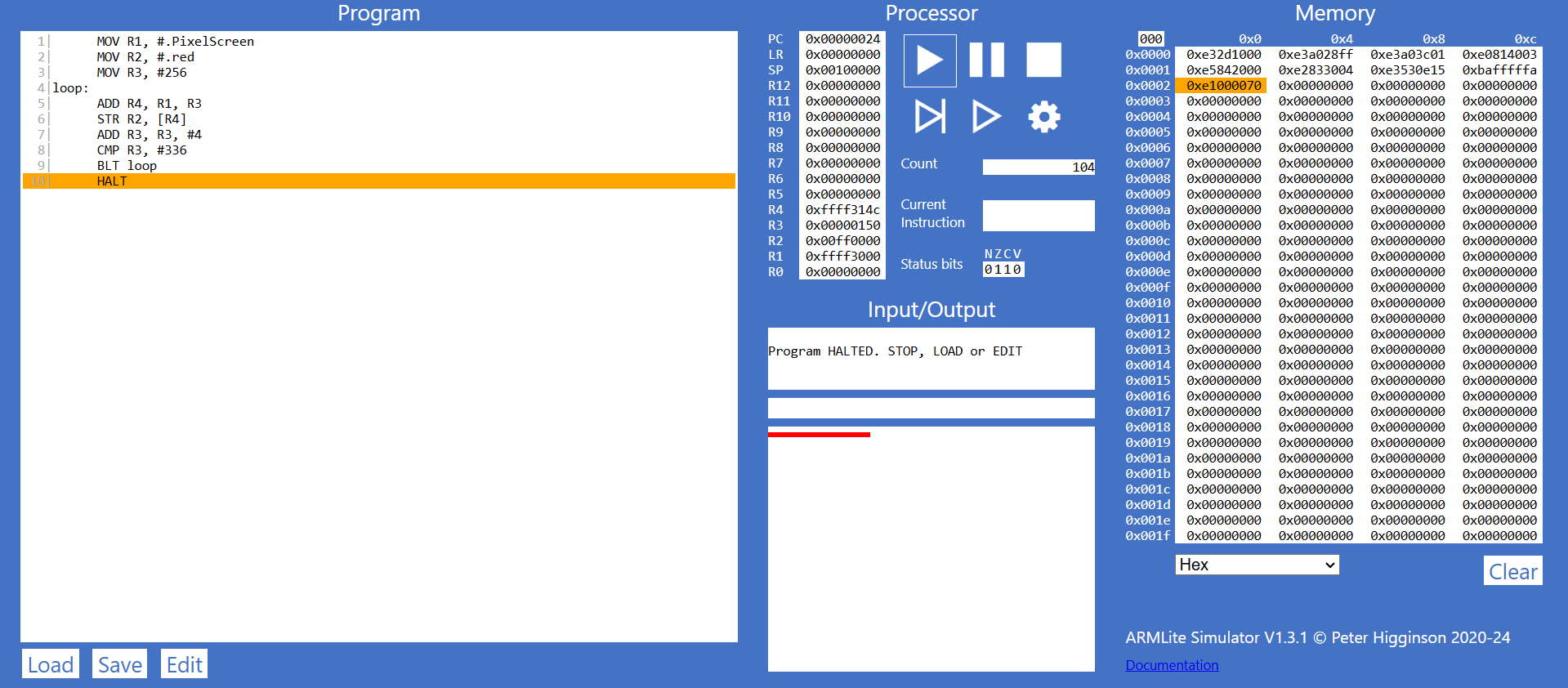
- Base Address: The base address of the pixel display memory is stored in register R1 (MOV R1, #.PixelScreen).

- Offset Calculation: The offset (R3) is added to the base address to calculate the actual address of the pixel (ADD R4, R1, R3).

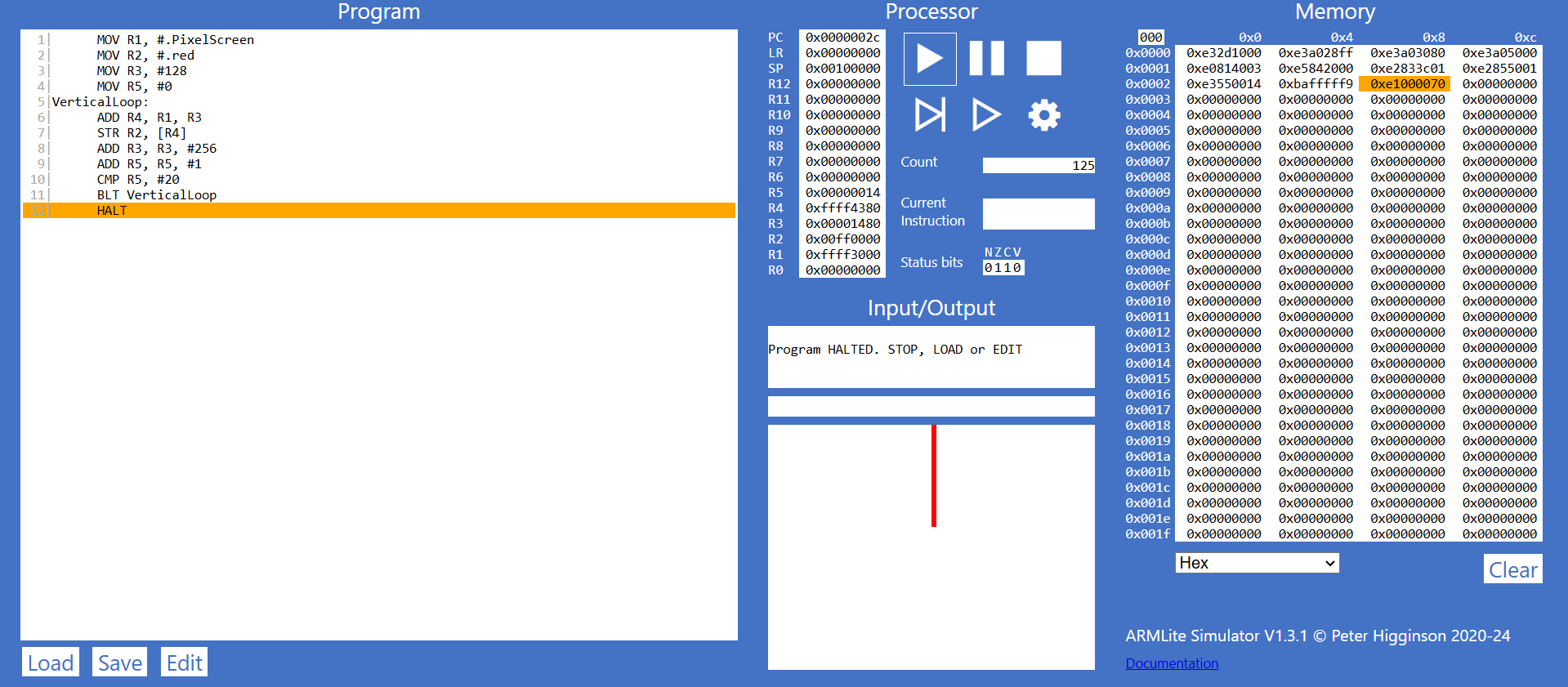
- Storing the Value: The red color value is stored at the calculated address (STR R2, [R4]).

This method allows the program to dynamically calculate the address of each pixel, making it an example of indirect addressing.

***(b)***

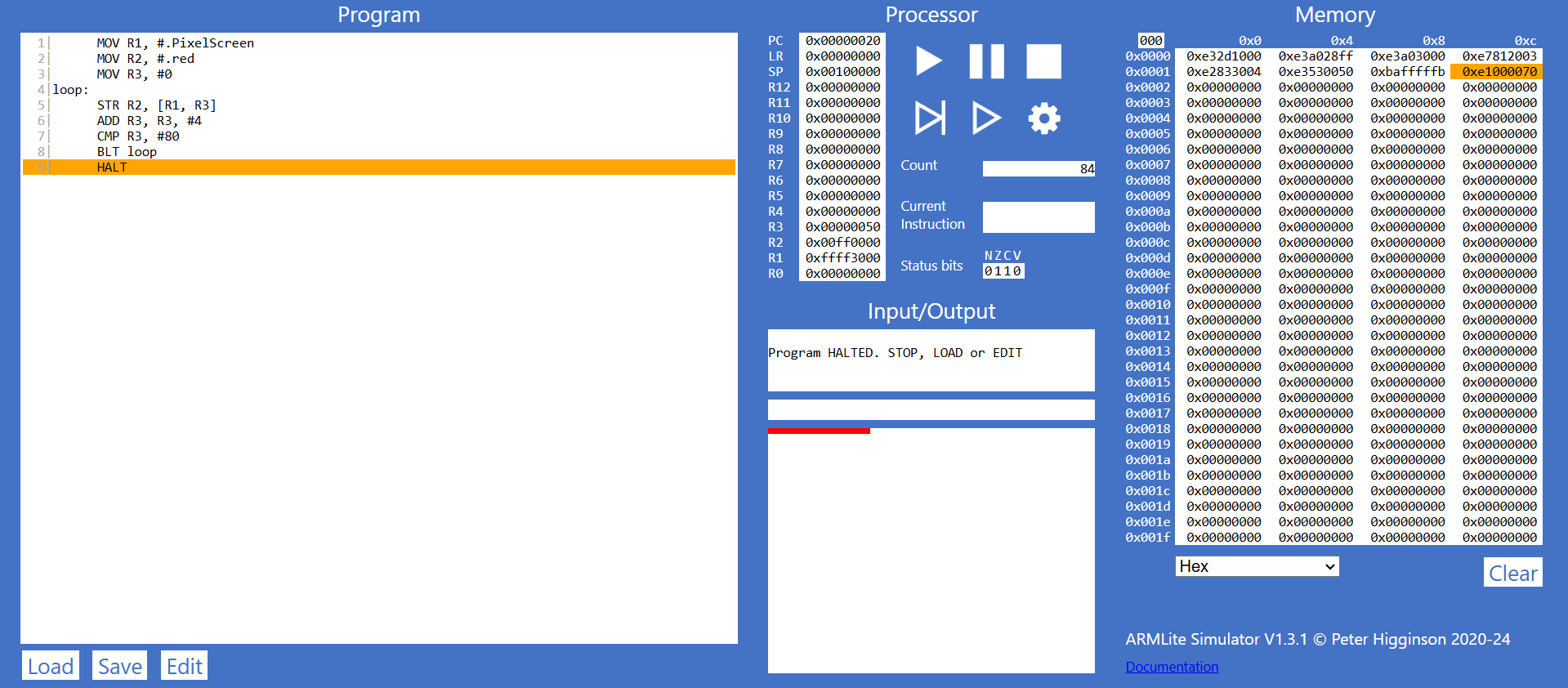


***(c)***

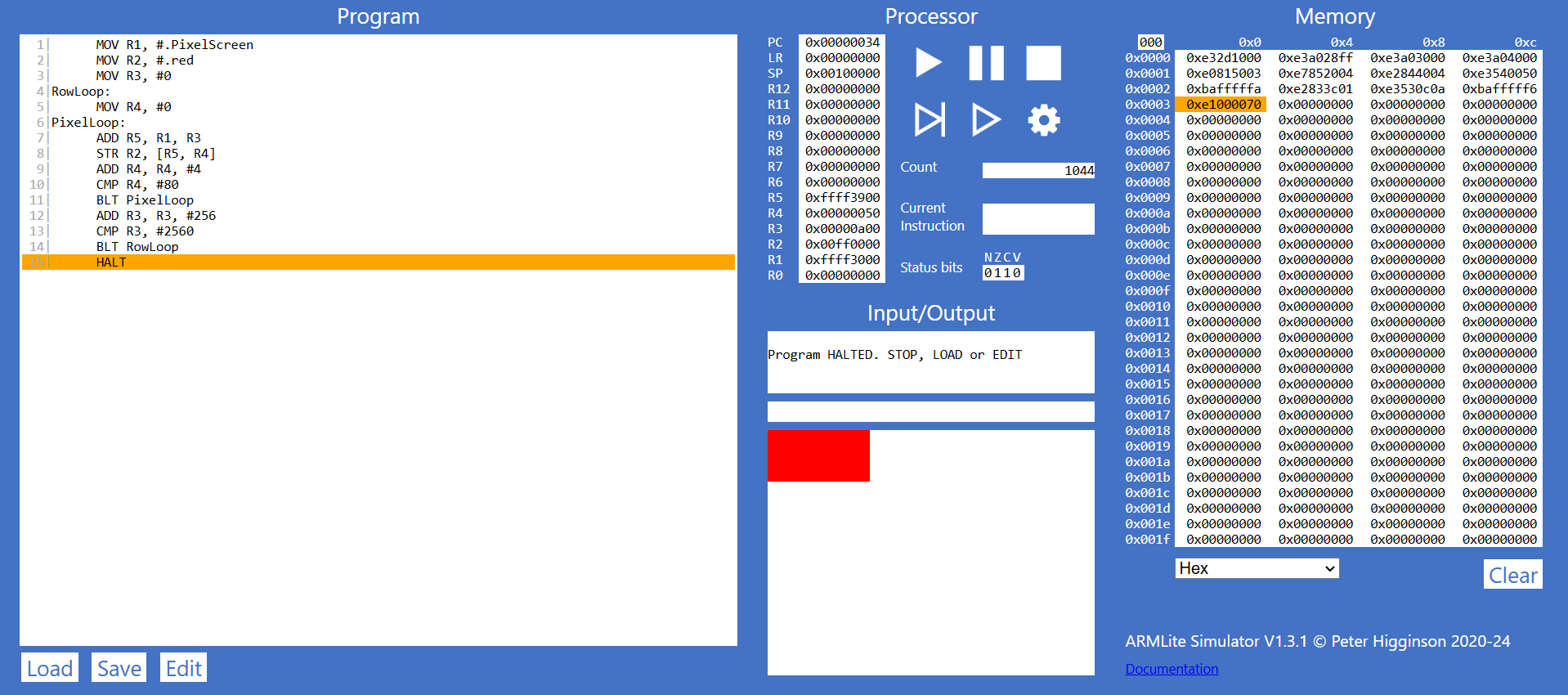


***Part 9.2***

***9.2.1***



***9.2.2***

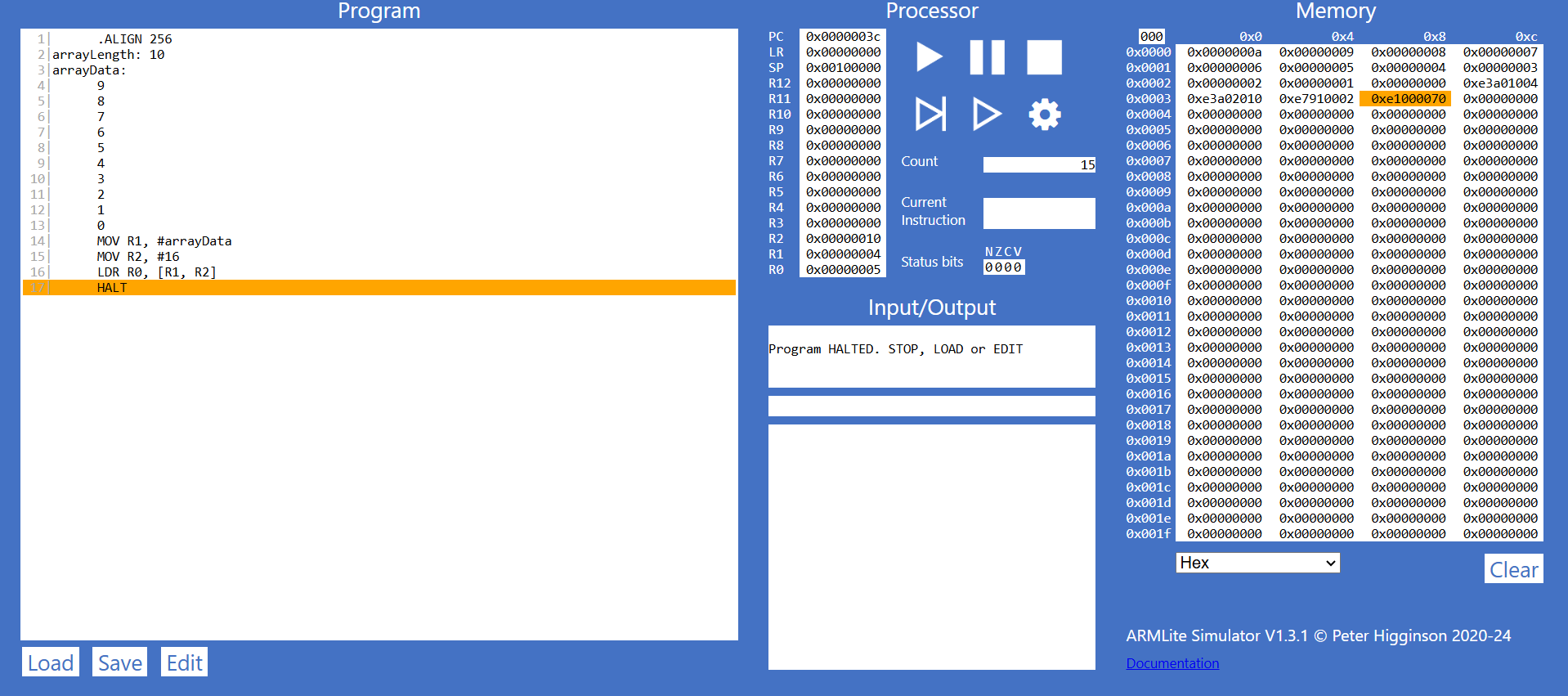


***Part 9.3***

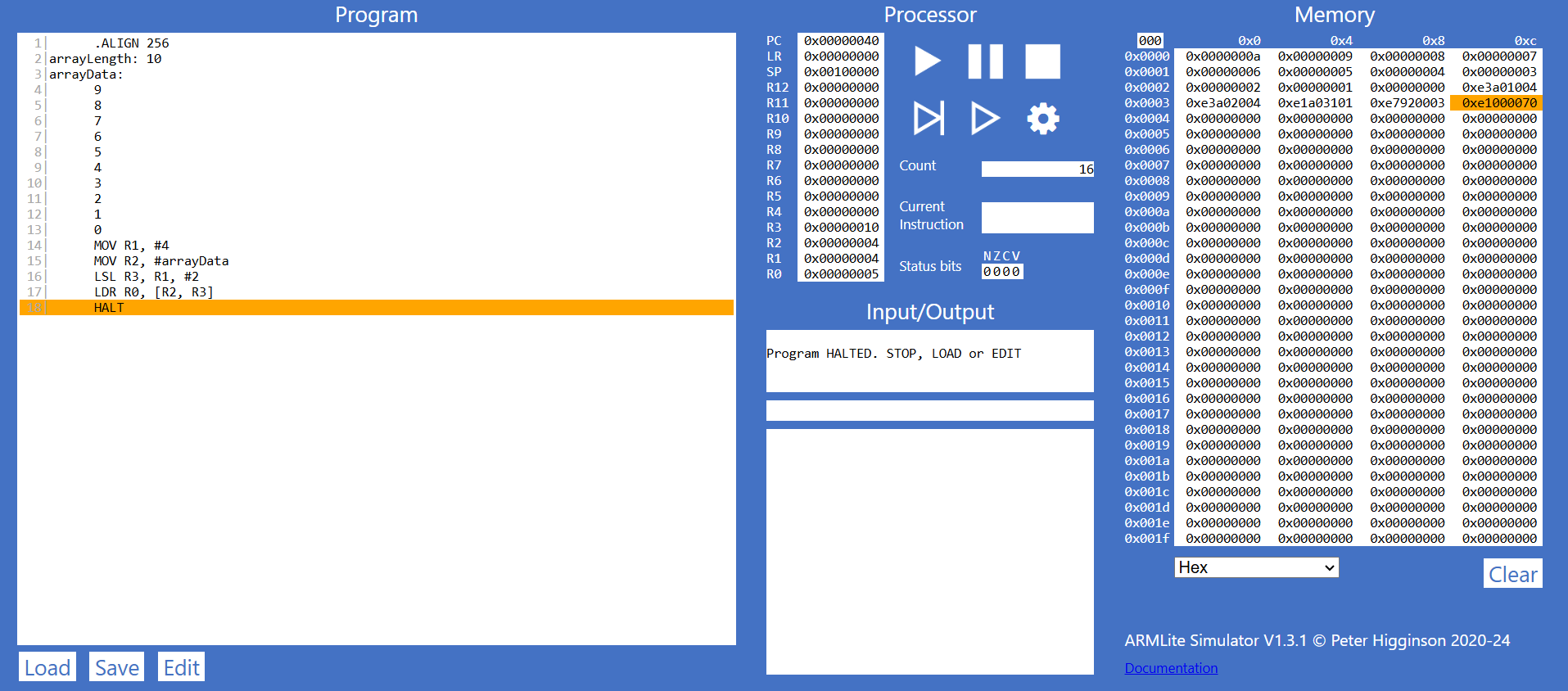
***9.3.1(a)***

==>The ”.ALIGN” 256 instruction in ARM assembly is used to align the following data on a 256-byte boundary, ensuring that the starting address of  ”arrayData” is a multiple of 256. This alignment can improve performance, as some processors access aligned memory more efficiently. It also helps maintain a consistent memory layout, which is useful for data processing and memory management, and ensures compatibility with hardware or software that may require specific alignment. In this case, aligning the array on a 256-byte boundary ensures it starts at a well-defined address, optimizing memory access patterns.

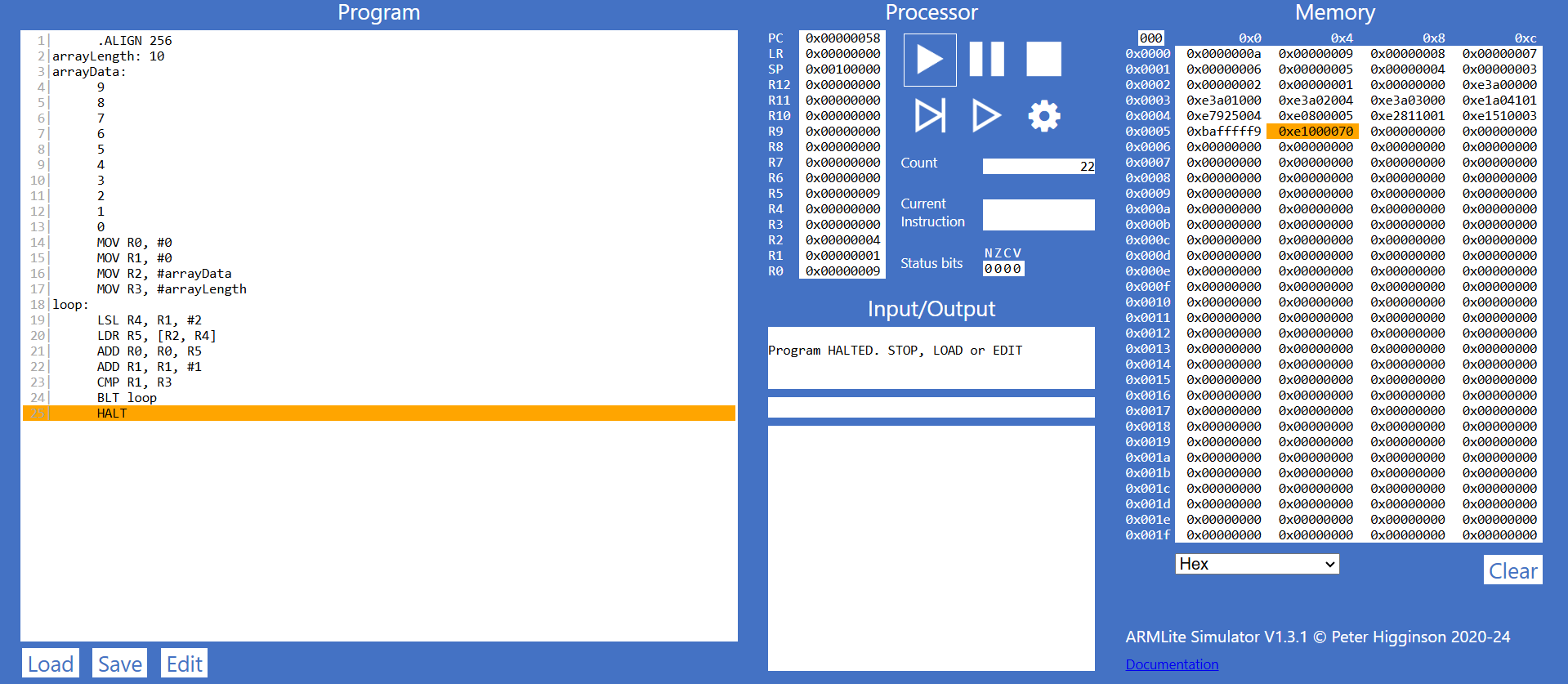
***9.3.1(b)***



***9.3.1(c)***



***9.3.2***



***Part 9.4***

***9.4.1***



***9.4.2***

