Name: Tran Duc Anh Dang

7.1.1 - On that specific memory, the value 0x00000065 is seen after entering the value 101. This is due to the fact that the number 101 in hexadecimal is represented as 65.

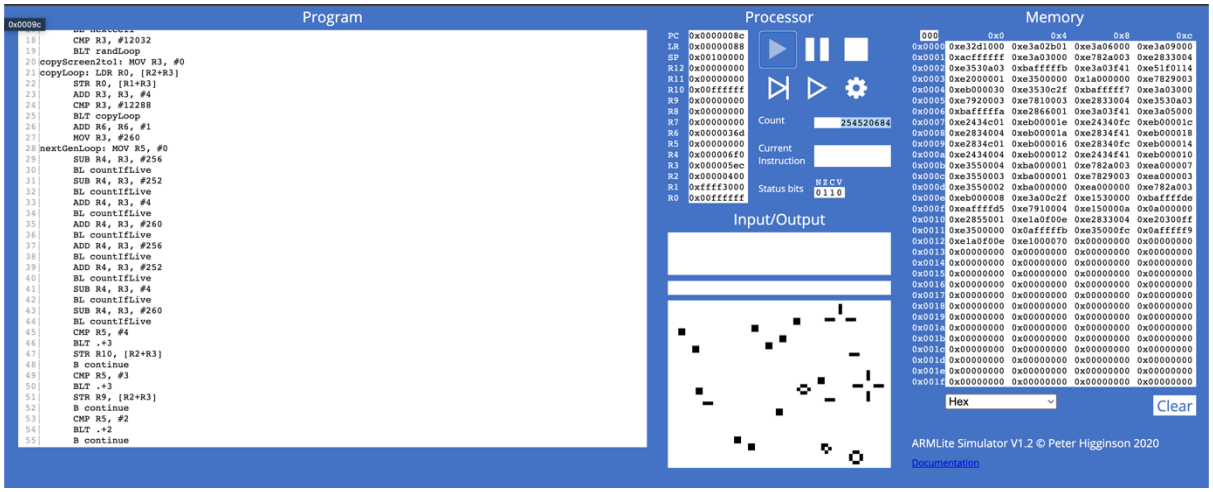
7.1.2 - The value 0x00000101 is shown on that memory after inputting the value 0x101 since 0x101 is represented in hexadecimal format, eliminating the requirement to convert it to hexa.

7.1.3 - The value 0b101 was shown after inputting it. Because 0b101 is represented in binary, it may be translated to 0x00000005 in hexadecimal.

7.1.4 - The rows and columns remain represented regardless of how the data is represented inside the memory box. I think it should be that way since altering how rows and columns are represented can end up requiring more space.

7.2.1 - Because each memory word needs 16 bits to store it. That why the memory address offset go up in multiples of 0x4. (bytes)

7.3



7.3.1 - Before loading the data to the memory grid to await execution, the assembler read the instruction and converted it into machine code.

7.3.2 - According to what I've learned, the hex value that hovers everytime I move the laptop mouse to the line number corresponds to the memory location where that instruction is kept.

7.3.3

- The extra spaces and blank lines won't be saved in the memory.
- After the code is submitted, the comments, whether they are on their own line or on the same line as the instruction, will be coloured.
- If a remark is added, the line count will go up. The rest will remain unchanged.
- The first line of code will display an error if the comma is changed.

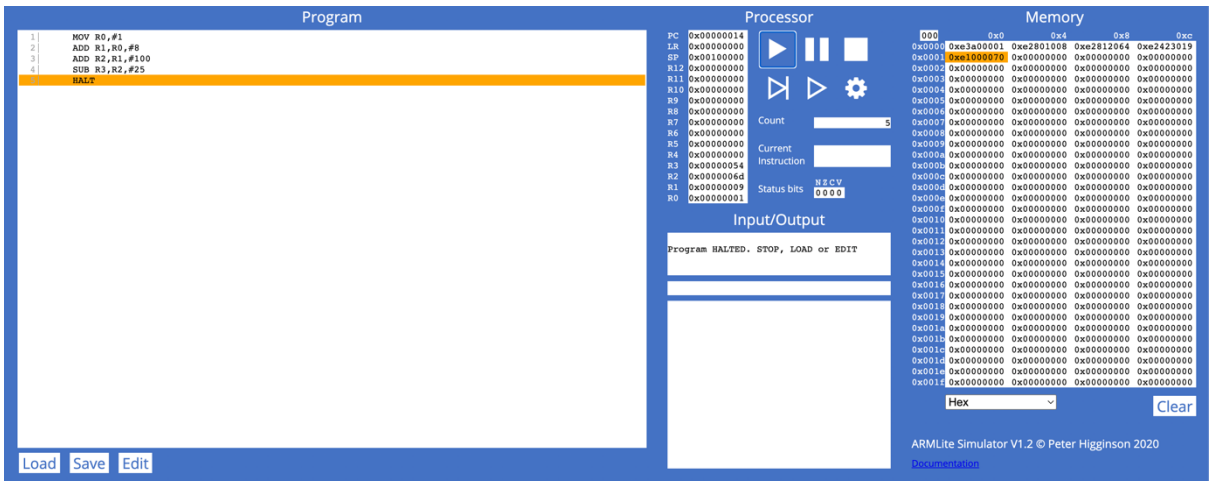
7.4.1 - The highlighted text on both screens indicates the line of code that will be run by the programme before it is halted.

7.4.2 - Every time we click the button that is highlighted in red, the programme will run line by line.

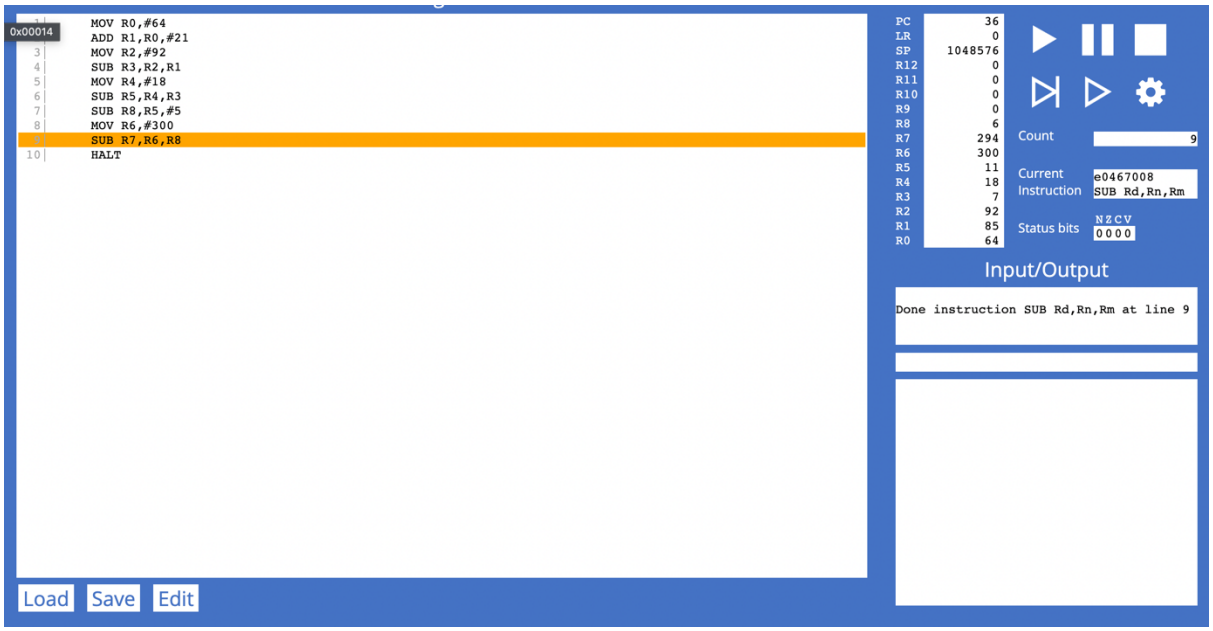
7.4.3 - The processor paused just before the line with the breakpoint.

7.5.1 - First, the instruction will load the value 1 into register 0. The value in R0 will then be added to one, and the result will be stored in reg01.

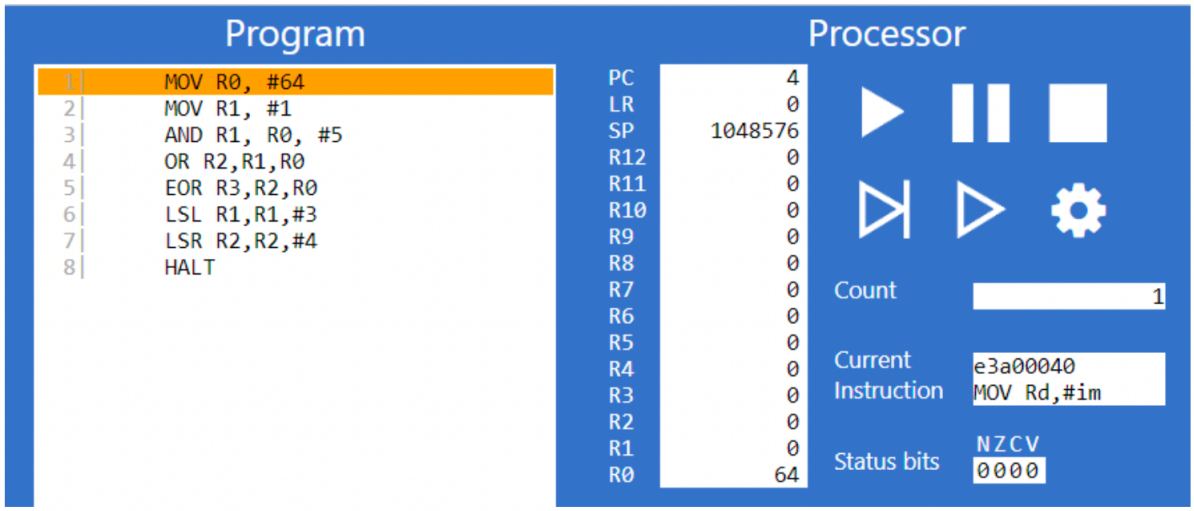
7.5.2



7.5.3

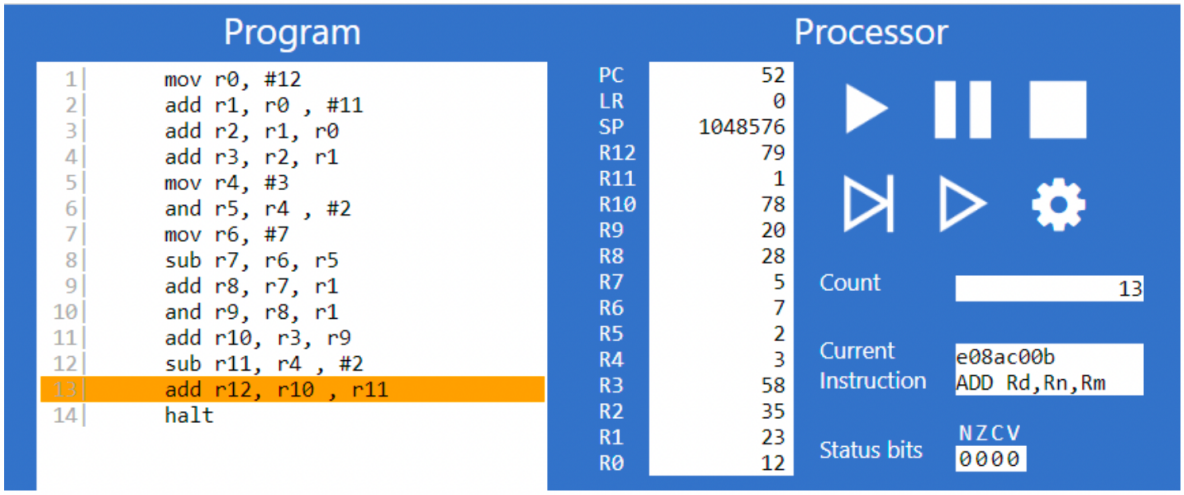


7.5.4



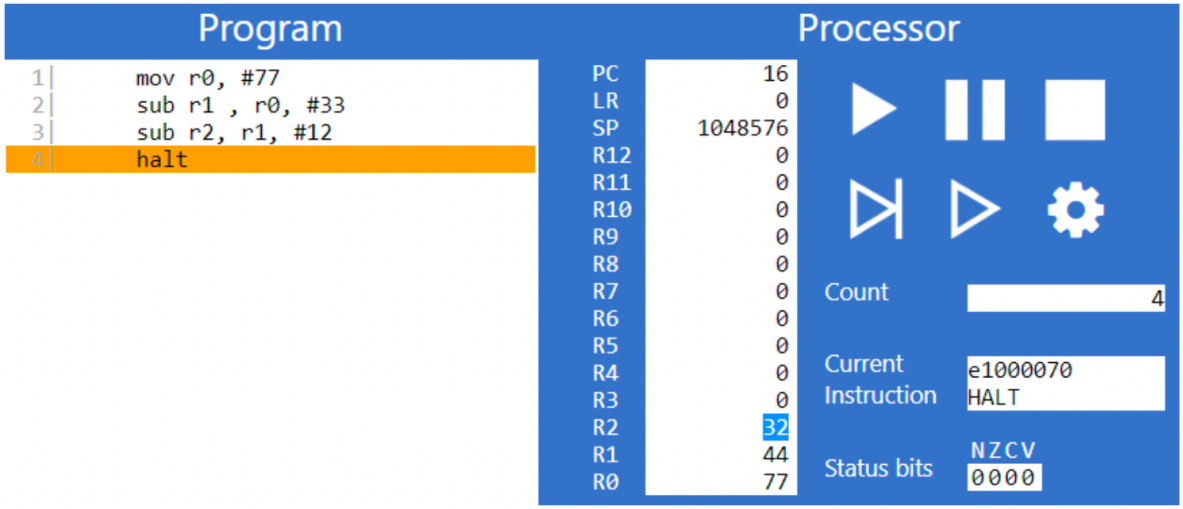
Instruction	***Decimal value of the destination register after executing this instruction***	***Binary value of the destination register after executing this instruction***
MOV R0, #64	64	01000000
MOV R1, #1	1	0000001
AND R1, R0, #5	0	0000000
OR R2,R1,R0	64	01000000
EOR R3,R2,R0	0	0000000
LSL R1,R1,#3	0	0000000
LSR R2,R2,#4	4	00000100

7.5.5



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	00001100
ADD R1, R0, #11	23	0010111
ADD R2,R1,R0	35	0100011
ADD R3 , R2, R1	58	0111010
MOV R4,#3	3	0000011
AND R5,R4,#2	2	0000010
MOV R6, #7	7	00000111
SUB R7,R6,R5	5	00000101
ADD R8,R7,R1	28	00011100
AND R9, R8, R1	30	00010100
ADD R10,R3,R9	78	01001110
SUB R11, R4, #2	1	00000001
ADD R12,R10,R11	79	01001111

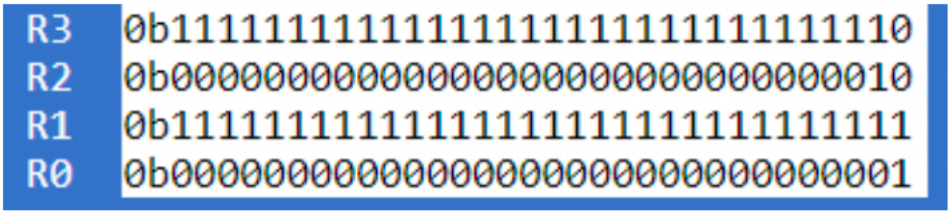
7.5.6



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 R1, R0, #33	44	101100
SUB R2,R1,#12	32	100000

7.6.1 - Because the value is represent in signed decimal

7.6.3



The pattern is to flip a negative integer from its positive equivalent in binary, then add 1.

7.6.4

This negative integer is changed to its positive value, which is then saved in r1.