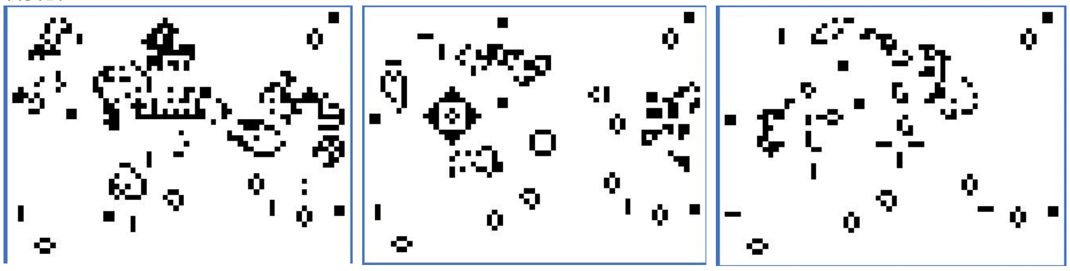
Student's ID: 104209393
Student's name: Ai Vi Tran

7.1.1: 65 (it changes from decimal to hexadecimal)
7.1.2: 0x101 (the input has already been hex so there are no changes)
7.1.3: 5 (it changes from binary to hex)
7.1.4: No

7.2.1: Because each of the memory word consists of 32 bits or 4bytes. Therefore, when we want to access a word in memory, we need to specify the address of the first byte of that word. Since each byte in the memory has a unique address, the address for each consecutive word in memory will differ by four bytes (0x4) from each other

7.3.1:



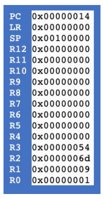
7.3.2:
Assemblers are programs that convert assembly language into machine code that can be executed by a computer's processor. In this case, the code that was entered into the ARMlite simulator was assembly language code, which was then assembled by the simulator into machine code. The Von Neumann architecture refers to the design of modern computers, where both data and instructions are stored in the same memory and accessed through the same bus.
When the code was submitted, the simulator assembled the code and loaded it into the memory, which is why the memory window changed and there were values in the first 13 rows of memory words. The line numbers added by ARMlite are simply a way to help navigate and discuss the code. The pop-up tooltip showing a 5 digit hex value when hovering over a line of code is simply displaying the memory address where the corresponding machine code for that line of assembly language is stored.
7.3.3:
The 5-digit hex value that appears when hovering over a line number in the source code represents the memory address of the corresponding instruction or data in the ARMlite's memory. Each line in the source code is translated into one or more instructions, and each instruction is loaded into memory at specific address. The line number is just a reference point to help navigate and debug the code, but the actual execution of the code is based on the memory addresses.

- The blank lines, additional spaces disappear
- The line numbers change
- The comments turn green
- The totel number of instructions remains the same

7.4.1: That show the input instruction, and the address of it in memory
7.4.2: It will execute the next instruction in the program and then pause execution again, it is useful for debugging and understanding how the program works.

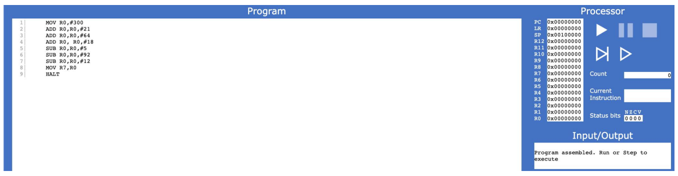
7.5.1:
1) Move number 1 to register 0
2) add 8 to number in register 0 and put the result in register 1
3) add 100 to the number in register 1 and put the result in register 2
4) The number in register 2 minus 25 and put the result in register 3
5) End the program

7.5.2:



```
PC  0x00000014
LR  0x00000000
SP  0x00100000
R12 0x00000000
R11 0x00000000
R10 0x00000000
R9  0x00000000
R8  0x00000000
R7  0x00000000
R6  0x00000000
R5  0x00000000
R4  0x00000000
R3  0x00000054
R2  0x0000006d
R1  0x00000009
R0  0x00000001
```

7.5.3:



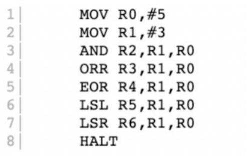
```
        MOV R0,#300
        ADD R0,R0,#21
        ADD R0,R0,#64
        ADD R0, R0,#18
        SUB R0,R0,#5
        SUB R0,R0,#92
        SUB R0,R0,#12
        MOV R7,R0
        HALT
```

```
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
Status bits NZCV 0000

Program assembled. Run or Step to
execute
```

7.5.4:



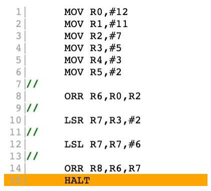
```
        MOV R0,#5
        MOV R1,#3
        AND R2,R1,R0
        ORR R3,R1,R0
        EOR R4,R1,R0
        LSL R5,R1,R0
        LSR R6,R1,R0
        HALT
```

Instruction	Decimal value of the destination register after executing this instruction	Binary value of the destination register after executing this instruction

MOV	R0 = 5
	R1 = 3
		R0 = 0000 0101
		R1 = 0000 0011

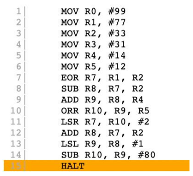
AND	R2 = 1	R2 = 0000 0001
ORR	R3 = 7	R3 = 0000 0111
EOR	R4 = 6	R4 = 0000 0110
LSL	R5 = 96	R5 = 011 00000
LSR	R6 = 0	R6 = 0000 0000

7.5.5:



```
        MOV R0,#12
        MOV R1,#11
        MOV R2,#7
        MOV R3,#5
        MOV R4,#3
        MOV R5,#2
//
        ORR R6,R0,R2
//
        LSR R7,R3,#2
//
        LSL R7,R7,#6
//
        ORR R8,R6,R7
        HALT
```

7.5.6:



```
        MOV R0, #99
        MOV R1, #77
        MOV R2, #33
        MOV R3, #31
        MOV R4, #14
        MOV R5, #12
        EOR R7, R1, R2
        SUB R8, R7, R2
        ADD R9, R8, R4
        ORR R10, R9, R5
        LSR R7, R10, #2
        ADD R8, R7, R2
        LSL R9, R8, #1
        SUB R10, R9, #80
        HALT
```

7.6.1: The use of the LSL instruction to perform a left shift on the value in R0 causes the most significant bit of R1 to be set to 1, which indicates that the number is negative.

7.6.3: