COS10004 – Computer System
Name: Phan Vũ – Student Id: 104222099
LAB 7

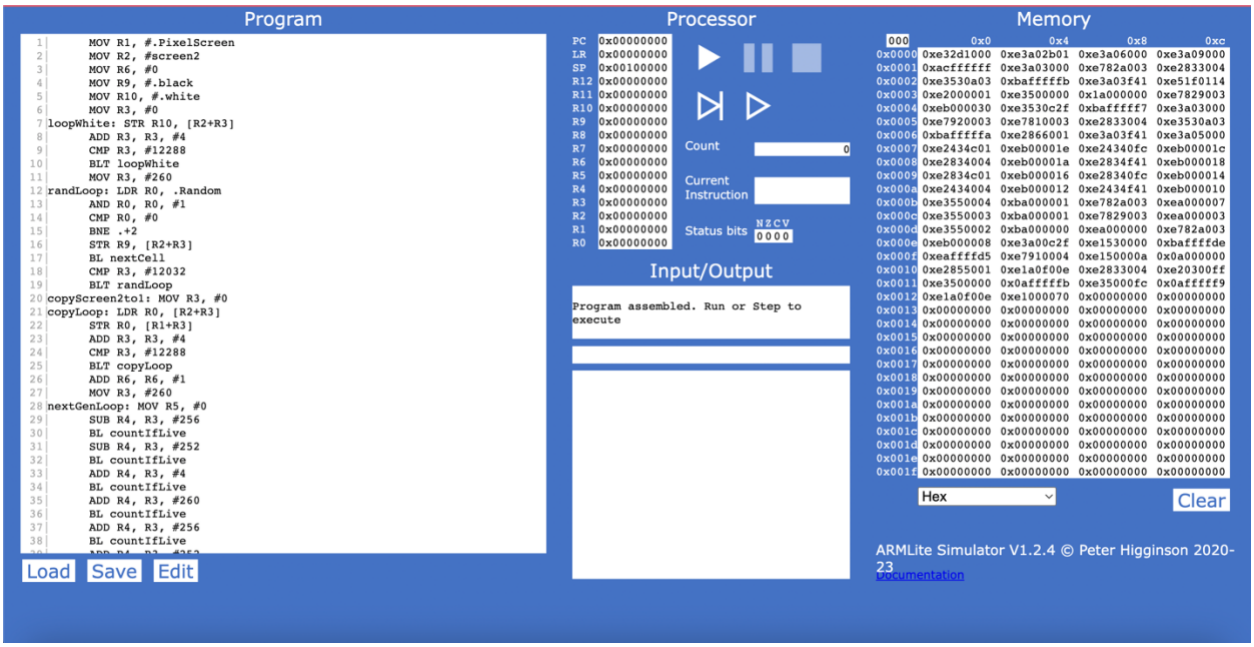
7.1.1 What value is displayed? Why?
0x00000065
Because 101 in hex is 65, the 0x is a standard prefix indicating that what follows is in hexadecimal (hex) format.

7.1.2 What value is displayed, and why?
0x00000101
Because it is already in hex format.

7.1.3 What value is displayed, and why?
0x00000005
Because it is 5 in hex, the tooltip tells us each 32-bit word. You can see them in binary and in 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?
Yes. The row header shows the first four digits of the address, and the full address is specified by appending the single hex digit shown in the column header. Thus, 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?
Each word of 32 bits is made up of four 8-bit 'bytes
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.

First, it 'assembled' (translated) the assembly language into machine code; then it loaded the machine code into memory.

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?

It indicates the address in the memory of the corresponding machine code instruction.

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

Where the code has paused, it indicates the address in memory of the corresponding machine code instruction.

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

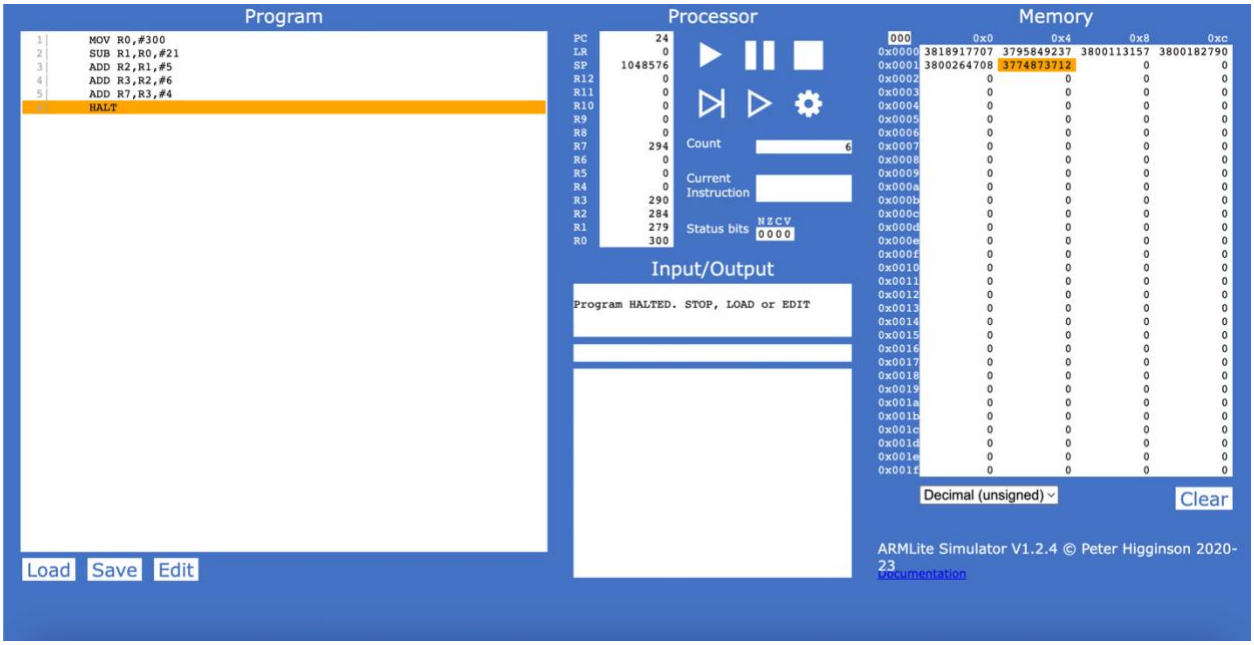
In the case of the red circled button, these two buttons allow you to slow things down to literally single steps of code execution. This is invaluable for debugging code, mainly when you want to check whether the outcome of a given instruction has produced what you expect (be that a value in memory, in a register, or a graphical element on display etc.).

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

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?
This program consists of five lines, each with a single instruction. Each instruction consists of a "operation," which is presented here in "mnemonic" form (typically an abbreviation of the instruction's description). MOV is the operation to move a value (which actually means 'copy and move'). You probably guessed that ADD and SUB are the operations to add and subtract values, respectively, and that HALT halts the program's execution. Except for HALT, each operation is followed by up to three 'operands' that specify what the operation is applied to in each instance. When multiple operands are present, they must be separated by commas. The final operand in the first four instructions consists of one of our initial numbers: 1,8,100 and 25, preceded by the # symbol (pronounce 'hash', not 'hashtag'). These are known as 'immediate' values in assembly language programming, as they are written directly into the program code. The remaining operands, R0, R1, R2, and R3, designate registers used to store initial values, intermediate calculations, and the final result.

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

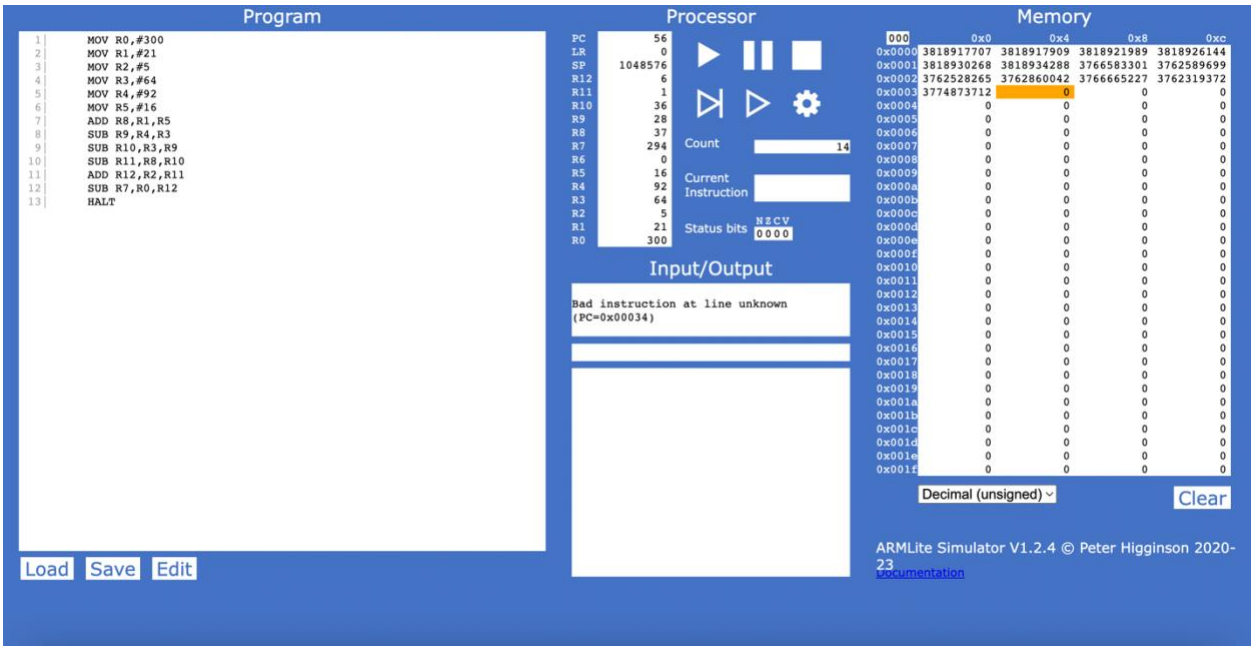
7.5.3 Task: Your 6 initial numbers are now 300, 21, 5, 64, 92, and 18. Write an Assembly Program that uses these values to compute a final value of 294 (you need only use MOV, ADD and SUB). Place your final result in register R7(don't forget the HALT instruction) When the program is complete, take a screenshot of the code and the register table.



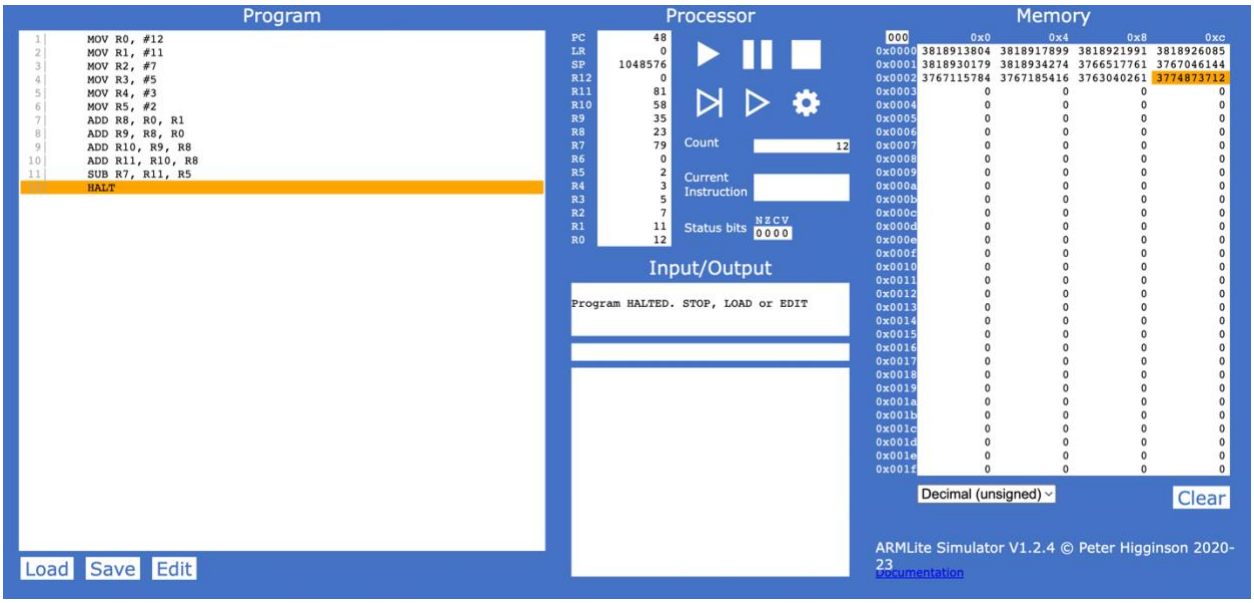
7.5.4 Task: Write your own simple program, that starts with a MOV (as in the previous example) followed by five instructions, using each of the five new instructions listed above, once only, but in any order you like – plus a HALT at the end, and with whatever immediate values you like.

Instruction	Decimal value of the destination register after executing this instruction	Binary value of the destination register after executing this instruction
MOV R0, #14	3818913806	0b11100011101000000000000000001110
STR R0, 20	3851354120	0b11100101100011110000000000001000
HALT	3774873712	0b11100001000000000000000001110000
LDR R1, 76	3852406840	0b11100101100111110001000000111000
HALT	3774873712	0b11100001000000000000000001110000

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).



When the program is complete, take a screen shot of the code and the register table and paste into your submission document.



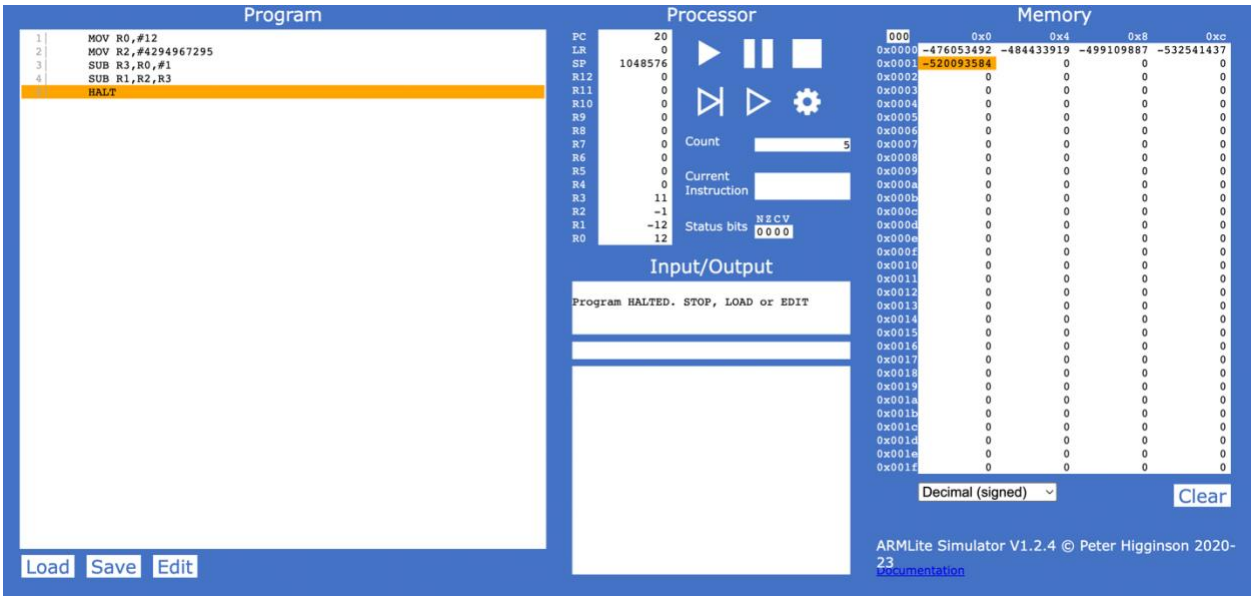
7.5.6: Let's play again !

When the program is complete, take a screen shot of the code and the register table and paste into your submission document.

7.6.1 - Why is the result shown in R1 a negative decimal number and with no obvious relationship to 9999?
Because the "LSL R1,R0,#18" command is to add 18 digits 0 at the end of R0 value (in binary form) and assign the value to R1, so R1 has the value that has no relationship to 9999 (which is 10011100001111 in binary)

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.
For 1: #1
For -1: #4294967295 (It's $2^{32} - 1$)
For 2: #2
For -2: #4294967294 (It's $2^{32} - 2$)

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. Take a screen shot showing your program and the registers after succesful execution, and paste into your submission document


Program
1 MOV R0,#12
2 MOV R2,#4294967295
3 SUB R3,R0,#1
4 SUB R1,R2,R3
5 HALT
Load
Save
Edit
Processor
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 11
R2 -1
R1 -12
R0 12
Count 5
Current Instruction
Status bits NZCV 0000
Input/Output
Program HALTED. STOP, LOAD or EDIT
Memory
000 0x0 0x4 0x8 0xc
0x0000 -476053492 -484433919 -499109887 -532541437
0x0001 -520093584 0 0 0
0x0002 0 0 0 0
0x0003 0 0 0 0
0x0004 0 0 0 0
0x0005 0 0 0 0
0x0006 0 0 0 0
0x0007 0 0 0 0
0x0008 0 0 0 0
0x0009 0 0 0 0
0x000a 0 0 0 0
0x000b 0 0 0 0
0x000c 0 0 0 0
0x000d 0 0 0 0
0x000e 0 0 0 0
0x000f 0 0 0 0
0x0010 0 0 0 0
0x0011 0 0 0 0
0x0012 0 0 0 0
0x0013 0 0 0 0
0x0014 0 0 0 0
0x0015 0 0 0 0
0x0016 0 0 0 0
0x0017 0 0 0 0
0x0018 0 0 0 0
0x0019 0 0 0 0
0x001a 0 0 0 0
0x001b 0 0 0 0
0x001c 0 0 0 0
0x001d 0 0 0 0
0x001e 0 0 0 0
0x001f 0 0 0 0
Decimal (signed)
Clear
ARMLite Simulator V1.2.4 © Peter Higginson 2020-23
Documentation