Linnæus University
Sweden

Lab Assignment 1

LEGv8 Assembly Programming

Computer Technology 1

September 20, 2025

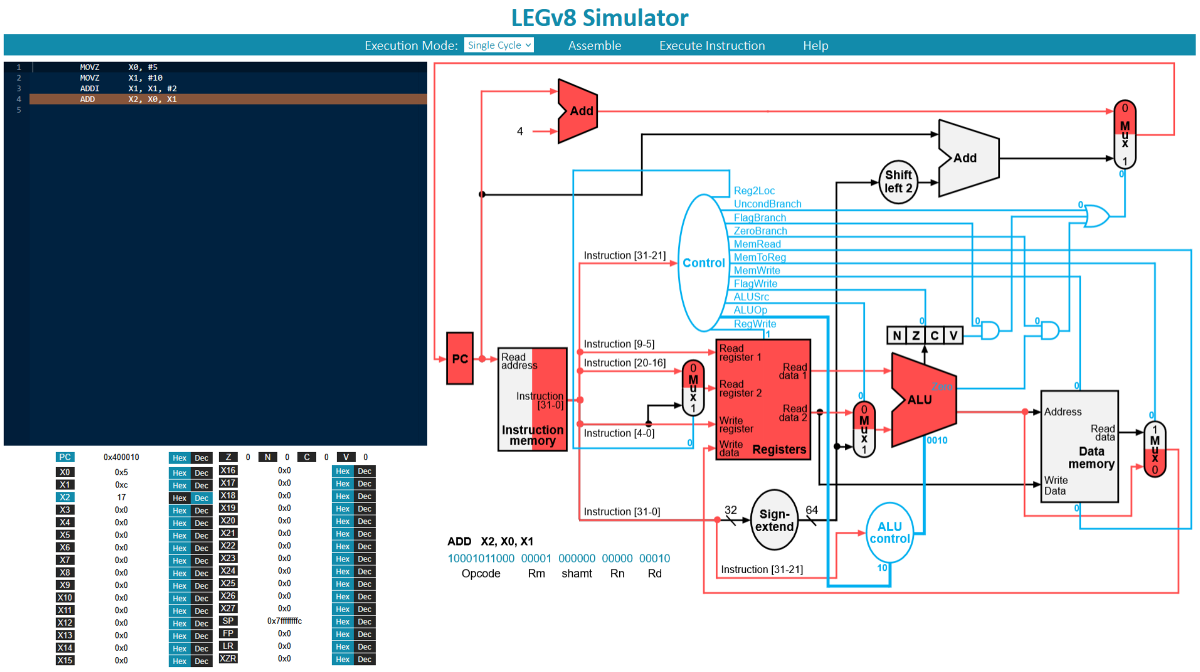
Author: Michelle Weber,
Sanja Janevska
Examiner: Mehdi Saman Azari
Semester: HT25
Discipline: NGDNS, NGDPV
Course code: 1dt301

Task 1 – Basic Instructions in LEGv8

Write the following code in the simulator and run it: (Assemble and execute all instructions)

Listing 1: LEGv8 program for Task 1

```
MOVZ X0, #5
MOVZ X1, #10
ADDI X1, X1, #2
ADD X2, X0, X1
```



```
MOVZ      X0, #5
MOVZ      X1, #10
ADDI      X1, X1, #2
ADD       X2, X0, X1
```

PC	0x400010	Z 0	N 0	C 0	V 0

X0	0x5	X16	0x0
X1	0xc	X17	0x0
X2	17	X18	0x0
X3	0x0	X19	0x0
X4	0x0	X20	0x0
X5	0x0	X21	0x0
X6	0x0	X22	0x0
X7	0x0	X23	0x0
X8	0x0	X24	0x0
X9	0x0	X25	0x0
X10	0x0	X26	0x0
X11	0x0	X27	0x0
X12	0x0	SP	0x7ffffffffc
X13	0x0	FP	0x0
X14	0x0	LR	0x0
X15	0x0	XZR	0x0

```
ADD   X2, X0, X1
10001011000  00001  000000  00000  00010
Opcode        Rm     shamt    Rn     Rd
```

Answer

The number stored in register X2 is: 17

Linnæus University
Sweden

Task 2 – Machine Code to LEGv8 Assembly

Instruction 1

Machine Code: `11010010100000000001000000000010`

Assembly:

```
    MOVZ X2, #128
```

Explanation: Move and store the decimal value 128 in X2.

Instruction 2

Machine Code: `11010010100000000001110011100100`

Assembly:

```
    MOVZ X4, #231
```

Explanation: Move and store the decimal value 231 in X4.

Instruction 3

Machine Code: `11001011000000100000000010000101`

Assembly:

```
    SUB X5, X4, X2
```

Explanation: Subtract X4 from X2 and store the result in X5.

Instruction 4

Machine Code: `D360 0CA5`

`(translates to:  1101001101100000 0000110010100101)`

Assembly:

```
   LSL X5, X5, X0, #3
```

Explanation: Shift X5 by X0 and shift the result by another decimal value of 3, then store the result in X5. You could also add X0 and the decimal value of 3 together and then shift X5 by that result, so X5 « (X0 + 3).

Task 3 - LEGv8 Arithmetic

Create a LEGv8 Assembly program to calculate the value of the following expression:

$$4 \cdot 5 + 16 \cdot 11 + 25$$

When finished, the result shall be stored in register x0.

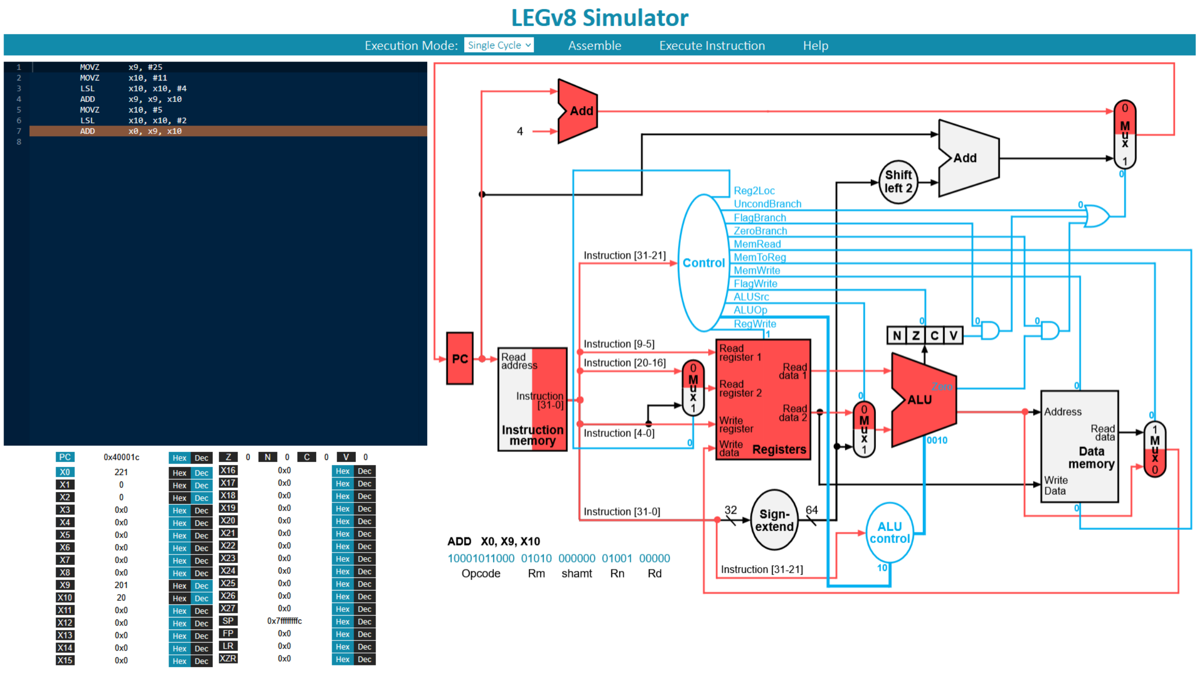
Code

Listing 2: LEGv8 program for Task 3

```
MOVZ x9, #25
MOVZ x10, #11
LSL x10, x10, #4
ADD x9, x9, x10
MOVZ x10, #5
LSL x10, x10, #2
ADD x0, x9, x10
```

Linnæus University
Sweden

Screenshot



```
MOVZ      x9, #25
MOVZ      x10, #11
LSL       x10, x10, #4
ADD       x9, x9, x10
MOVZ      x10, #5
LSL       x10, x10, #2
ADD       x0, x9, x10
```

```
ADD  X0, X9, X10
10001011000  01010  000000  01001  00000
  Opcode      Rm     shamt   Rn     Rd
```

```
PC   0x40001c   Hex Dec   Z 0   N 0   C 0   V 0
X0   221        Hex Dec   X16  0x0          Hex Dec
X1   0          Hex Dec   X17  0x0          Hex Dec
X2   0          Hex Dec   X18  0x0          Hex Dec
X3   0x0        Hex Dec   X19  0x0          Hex Dec
X4   0x0        Hex Dec   X20  0x0          Hex Dec
X5   0x0        Hex Dec   X21  0x0          Hex Dec
X6   0x0        Hex Dec   X22  0x0          Hex Dec
X7   0x0        Hex Dec   X23  0x0          Hex Dec
X8   0x0        Hex Dec   X24  0x0          Hex Dec
X9   201        Hex Dec   X25  0x0          Hex Dec
X10  20         Hex Dec   X26  0x0          Hex Dec
X11  0x0        Hex Dec   X27  0x0          Hex Dec
X12  0x0        Hex Dec   SP   0x7ffffffffc Hex Dec
X13  0x0        Hex Dec   FP   0x0          Hex Dec
X14  0x0        Hex Dec   LR   0x0          Hex Dec
X15  0x0        Hex Dec   XZR  0x0          Hex Dec
```

Explanation

We stored all mid-calculation results in the temporary storage. We first stored DEC(25) in register X9 and DEC(11) in register X10. Then, we calculated the square root of 16, which is 4. This way, we know what value to write in the LSL command. To shift values, you do multiplication with 2^{value}. So, to calculate $16 * 11$, we have to shift the value in X10 by 4. If we would calculate it manually, we would get $2^4 * 11 = 16 * 11 = 176$. Now, we can add that result to the value stored in X9, and store it in register X10. So we do: X9 + X10 or $176 + 25$. We now don't need the value stored in X10 anymore. We can safely store another value there, which will be DEC(5). Next, we need to find a way to multiply 4 with that value, that we stored in X10. This can again be done by finding the suqare root of that value. The suqare root of 4 is 2, so we shift the value in register X10 by 2 and store it in register X10. Now we have calculated $4 * 5$. Next, we just need to add both values from register X9 and X10 and store it in the desired register X0. The result is DEC(221).

Linnæus University
Sweden

Task 4 – Sum of Large Numbers

Instruction: Write a LEGv8 Assembly program to calculate the sum

$$1\,893\,423 + 443\,924$$

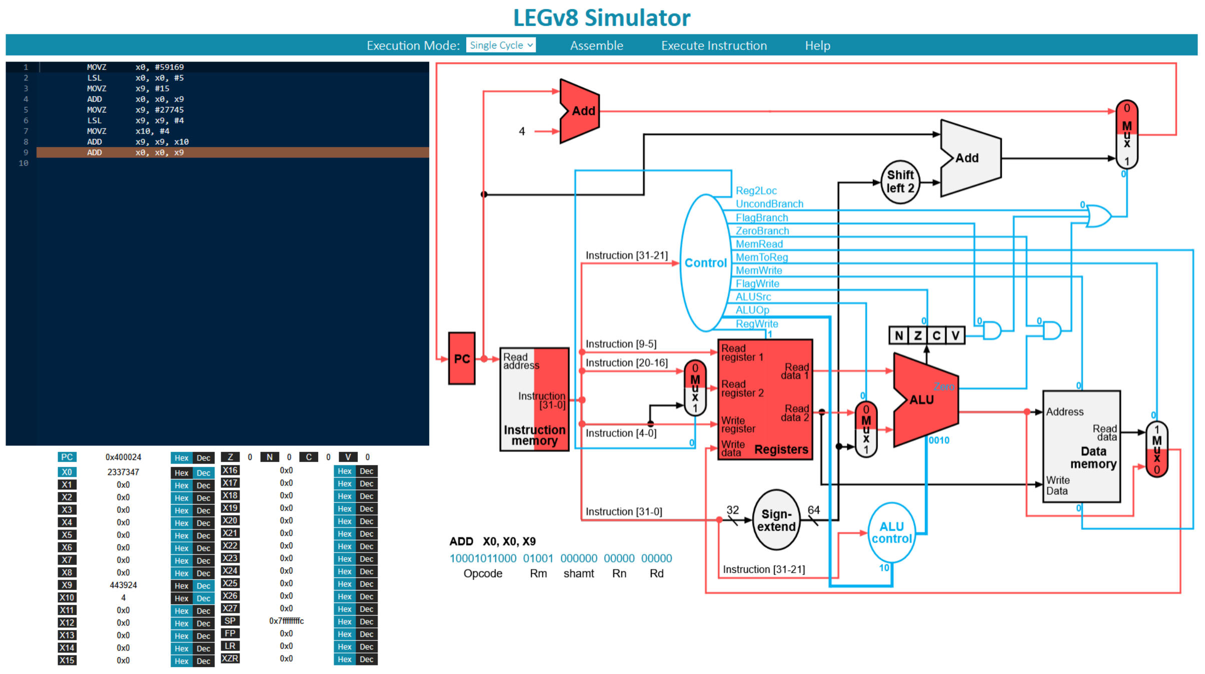
The numbers are decimal integers. You will probably encounter a problem to load these large numbers into registers, so you will have to find a way to solve this problem!

Code

Listing 3: LEGv8 program for Task 4

```
MOVZ x0, #59169
LSL x0, x0, #5
MOVZ x9, #15
ADD x0, x0, x9
MOVZ x9, #27745
LSL x9, x9, #4
MOVZ x10, #4
ADD x9, x9, x10
ADD x0, x0, x9
```

Screenshot



```
MOVZ      x0, #59169
LSL       x0, x0, #5
MOVZ      x9, #15
ADD       x0, x0, x9
MOVZ      x9, #27745
LSL       x9, x9, #4
MOVZ      x10, #4
ADD       x9, x9, x10
ADD       x0, x0, x9
```

```
PC    0x400024     Z 0   N 0   C 0   V 0
X0    2337347      X16   0x0
X1    0x0          X17   0x0
X2    0x0          X18   0x0
X3    0x0          X19   0x0
X4    0x0          X20   0x0
X5    0x0          X21   0x0
X6    0x0          X22   0x0
X7    0x0          X23   0x0
X8    0x0          X24   0x0
X9    443924       X25   0x0
X10   4            X26   0x0
X11   0x0          X27   0x0
X12   0x0          SP    0x7ffffffffc
X13   0x0          FP    0x0
X14   0x0          LR    0x0
X15   0x0          XZR   0x0
```

```
ADD   X0, X0, X9
10001011000  01001  000000  00000  00000
Opcode        Rm     shamt   Rn     Rd
```

Explanation

We tried to use the LSL command a lot and lower the numbers below $60\,000$, so they can be added without the simulator throwing errors. We split $1\,893\,423$ into $473\,355 * 2^2 + 3$ and $443\,924$ into $110\,981 * 2^2$. We always tried to lower the value as much as possible and remove some numbers if needed (residue). Kind of like integer division. We only want whole numbers, so the rest will be written as addition. $473\,355 * 2^2 + 3$ can be split further into $(236\,677 * 2^1 + 1)2^2 + 3$ and then into $((59\,169 * 2^2 + 1)2^1 + 1)2^2 + 3$, which can be combined into $59\,169 * 2^5 + 15$. The left number is now low enough to be loaded into the simulator. The right number still needs further splitting. $110\,981 * 2^2$ can be split into $110\,980 * 2^2 + 1$ and then into $(27\,745 * 2^2)2^2 + 1$, which can be combined into $27\,745 * 2^4 + 4$. Now the right side number is also low enough to be loaded into the simulator. So, we first move $59\,169$ into X0 and multiply the 2^5 by shifting the value by 5. We then store the value 15 in register X9, add it to X0 and store the result in

X0. So we now have $1\,893\,423$ stored in X0. Next, we store $27\,745$ into X9 and multiply it with 2^4 by shifting X9 by 4. Then, we need to add the value 4, by storing it in register X10 and then adding it to X9. The result will be stored in register X9. We now have $443\,924$ stored in X9. Lastly, we need to add these 2 values together, by adding X0 and X9 and storing the value in register X0. The final result of $2\,337\,347$ is now stored in register X0.

Task 5 – Sum of Odd Numbers

Write a LEGv8 Assembly program to calculate the sum

1 + 3 + 5 + ... + 99.

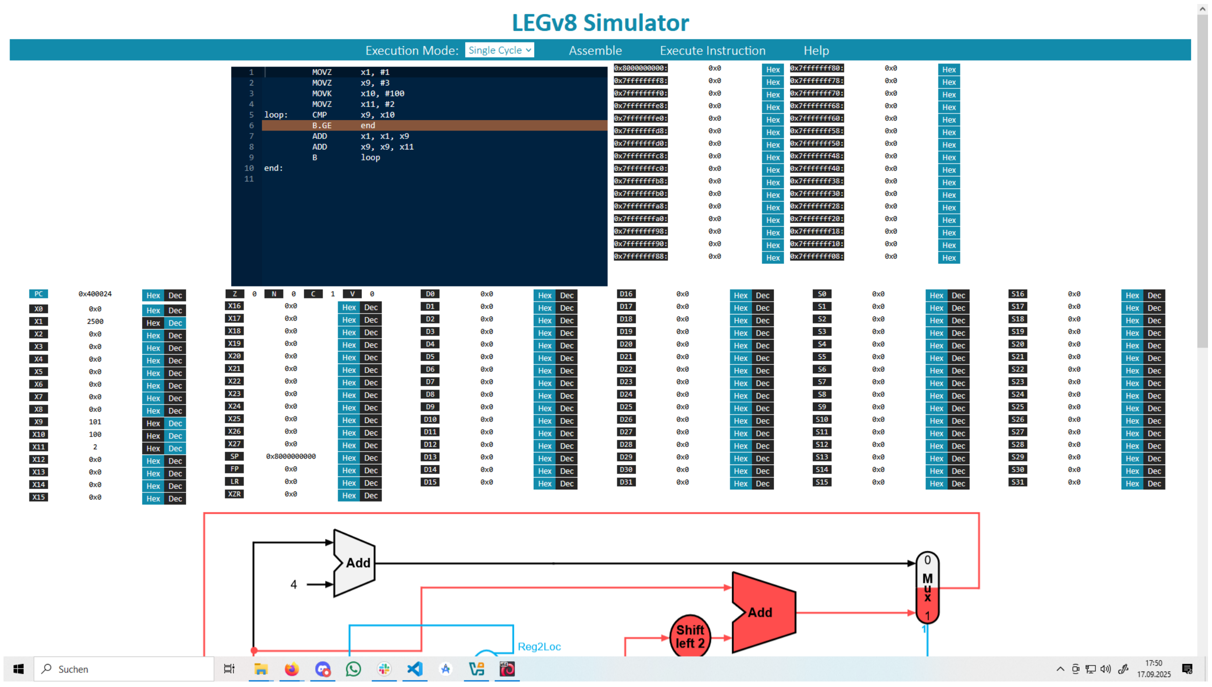
When finished, the sum shall be stored in register x1.

Code

Listing 4: LEGv8 program for Task 5

```
MOVZ x1, #1
MOVZ x9, #3
MOVK x10, #100
MOVZ x11, #2
loop: CMP x9, x10
      B.GE end
      ADD x1, x1, x9
      ADD x9, x9, x11
      B loop
end:
```

Screenshot



```
        MOVZ    x1, #1
        MOVZ    x9, #3
        MOVK    x10, #100
        MOVZ    x11, #2
loop:   CMP     x9, x10
        B.GE    end
        ADD     x1, x1, x9
        ADD     x9, x9, x11
        B       loop
end:
```

```
0x8000000000:  0x0    0x7fffffff80:  0x0
0x7ffffffff8:  0x0    0x7fffffff78:  0x0
0x7ffffffff0:  0x0    0x7fffffff70:  0x0
0x7fffffffe8:  0x0    0x7fffffff68:  0x0
0x7fffffffe0:  0x0    0x7fffffff60:  0x0
0x7fffffffd8:  0x0    0x7fffffff58:  0x0
0x7fffffffd0:  0x0    0x7fffffff50:  0x0
0x7fffffffc8:  0x0    0x7fffffff48:  0x0
0x7fffffffc0:  0x0    0x7fffffff40:  0x0
0x7fffffffb8:  0x0    0x7fffffff38:  0x0
0x7fffffffb0:  0x0    0x7fffffff30:  0x0
0x7fffffffa8:  0x0    0x7fffffff28:  0x0
0x7fffffffa0:  0x0    0x7fffffff20:  0x0
0x7fffffff98:  0x0    0x7fffffff18:  0x0
0x7fffffff90:  0x0    0x7fffffff10:  0x0
0x7fffffff88:  0x0    0x7fffffff08:  0x0
```

```
PC   0x400024      Z 0  N 0  C 1  V 0
X0   0x0           X16  0x0
X1   2500          X17  0x0
X2   0x0           X18  0x0
X3   0x0           X19  0x0
X4   0x0           X20  0x0
X5   0x0           X21  0x0
X6   0x0           X22  0x0
X7   0x0           X23  0x0
X8   0x0           X24  0x0
X9   101           X25  0x0
X10  100           X26  0x0
X11  2             X27  0x0
X12  0x0           SP   0x8000000000
X13  0x0           FP   0x0
X14  0x0           LR   0x0
X15  0x0           XZR  0x0

D0   0x0   D16  0x0   S0   0x0   S16  0x0
D1   0x0   D17  0x0   S1   0x0   S17  0x0
D2   0x0   D18  0x0   S2   0x0   S18  0x0
D3   0x0   D19  0x0   S3   0x0   S19  0x0
D4   0x0   D20  0x0   S4   0x0   S20  0x0
D5   0x0   D21  0x0   S5   0x0   S21  0x0
D6   0x0   D22  0x0   S6   0x0   S22  0x0
D7   0x0   D23  0x0   S7   0x0   S23  0x0
D8   0x0   D24  0x0   S8   0x0   S24  0x0
D9   0x0   D25  0x0   S9   0x0   S25  0x0
D10  0x0   D26  0x0   S10  0x0   S26  0x0
D11  0x0   D27  0x0   S11  0x0   S27  0x0
D12  0x0   D28  0x0   S12  0x0   S28  0x0
D13  0x0   D29  0x0   S13  0x0   S29  0x0
D14  0x0   D30  0x0   S14  0x0   S30  0x0
D15  0x0   D31  0x0   S15  0x0   S31  0x0
```

17:50
17.09.2025

Explanation

NOTE: Due to the given software being unable to accept any loop conditions and just stopping after not even looping once with ANY possible ending condition, we used another LEGv8-Simulator that was sent on slack. The link to that Simulator was on the github repository: https://simdeistud.github.io/LEGv8-Simulator/

We decided to loop the needed times, and each loop we will add 2 to the temporary register X9, which will represent the loop counter. The value 2 will be stored on the register X11 and will be added to X9 each loop. The max loop value is 100 (so $99 + 1$) and is stored on register X10. Each loop the registers X9 and X10 will be compared (CMP) and if the value stored on register X9 is greater or equal (B.GE) to X10, then the loop will end. So, this is the condition for the loop to end. Otherwise the loop will continue. The condition will be checked at the beginning of each loop. The main code of the loop consists of the addition of X1 and X9. Before the loop starts, we stored the starting value 1 on register X1 and the next value on register X9. So, in the first loop, those values will be added together and the result will be stored in register X1. With that, the first addition of $1 + 3$ is done. Next line in the code is to add the value 2 (from X11) to the register X9, which, again, is my loop counter. Since we dont need the "middle" loops, we can just skip them this way. By adding 2 instead of 1 and doing nothing in each odd loop, we also save time and performance. When the final loop is done, the result that is stored in register X1 is 2500.

Task 6 – Sum Numbers Stored in Memory

Instruction: Write a loop to add all the numbers stored in memory. When finished, the result shall be stored in register x0.

Memory Initialization (provided)

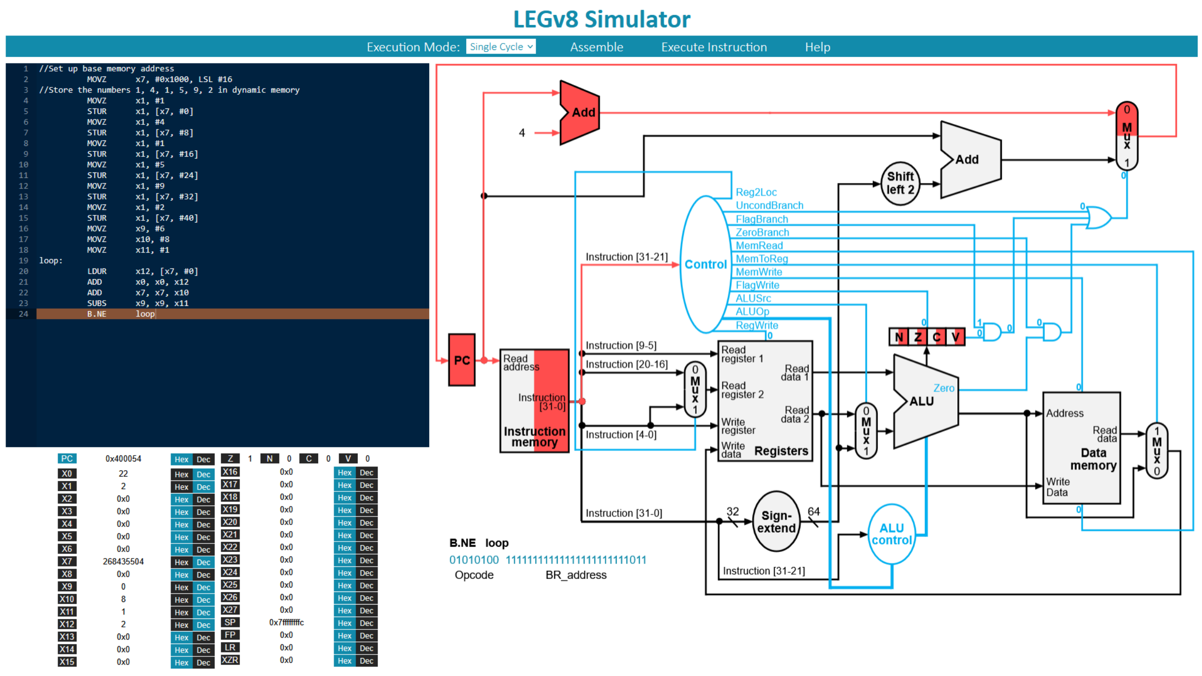
```
// Set up base memory address
MOVZ X7, #0x1000, LSL #16

// Store the numbers 1, 4, 1, 5, 9, 2 in memory
MOVZ x1, #1
STUR x1, [x7, #0]
MOVZ x1, #4
STUR x1, [x7, #8]
MOVZ x1, #1
STUR x1, [x7, #16]
MOVZ x1, #5
STUR x1, [x7, #24]
MOVZ x1, #9
STUR x1, [x7, #32]
MOVZ x1, #2
STUR x1, [x7, #40]
```

Code

Listing 5: LEGv8 program to sum memory values

```
    MOVZ x9, #6
    MOVZ x10, #8
    MOVZ x11, #1
    loop:
          LDUR x12, [x7, #0]
          ADD x0, x0, x12
          ADD x7, x7, x10
          SUBS x9, x9, x11
          B.NE loop
```

Screenshots



```
//Set up base memory address
           MOVZ      x7, #0x1000, LSL #16
//Store the numbers 1, 4, 1, 5, 9, 2 in dynamic memory
           MOVZ      x1, #1
           STUR      x1, [x7, #0]
           MOVZ      x1, #4
           STUR      x1, [x7, #8]
           MOVZ      x1, #1
           STUR      x1, [x7, #16]
           MOVZ      x1, #5
           STUR      x1, [x7, #24]
           MOVZ      x1, #9
           STUR      x1, [x7, #32]
           MOVZ      x1, #2
           STUR      x1, [x7, #40]
           MOVZ      x9, #6
           MOVZ      x10, #8
           MOVZ      x11, #1
loop:
           LDUR      x12, [x7, #0]
           ADD       x0, x0, x12
           ADD       x7, x7, x10
           SUBS      x9, x9, x11
           B.NE      loop
```

B.NE loop
01010100 111111111111111111111011
Opcode BR_address

PC	0x400054	Hex Dec	Z	1	N	0	C	0	V	0

X0	22	Hex Dec	X16	0x0	Hex Dec
X1	2	Hex Dec	X17	0x0	Hex Dec
X2	0x0	Hex Dec	X18	0x0	Hex Dec
X3	0x0	Hex Dec	X19	0x0	Hex Dec
X4	0x0	Hex Dec	X20	0x0	Hex Dec
X5	0x0	Hex Dec	X21	0x0	Hex Dec
X6	0x0	Hex Dec	X22	0x0	Hex Dec
X7	268435504	Hex Dec	X23	0x0	Hex Dec
X8	0x0	Hex Dec	X24	0x0	Hex Dec
X9	0	Hex Dec	X25	0x0	Hex Dec
X10	8	Hex Dec	X26	0x0	Hex Dec
X11	1	Hex Dec	X27	0x0	Hex Dec
X12	2	Hex Dec	SP	0x7ffffffffc	Hex Dec
X13	0x0	Hex Dec	FP	0x0	Hex Dec
X14	0x0	Hex Dec	LR	0x0	Hex Dec
X15	0x0	Hex Dec	XZR	0x0	Hex Dec

Explanation

We first decided to use the temoraries (X9 - X15) to store values that we need shortly to conduct the calculation. The list contained 6 numbers, so we need 6 loops total. Hence, we store the value 6 in register X9. STUR means that 64-bit values are getting stored, so 8 Bytes. Each value is stored every 8 Bytes, so the offset is 8. Therefore, we store the address of every value in register X10, which is 8 (the offset used here). Every loop, we remove 1 value from the list. In order to mark that in the loop counter and update the loopcounter properly, we subtract this value 1 each loop. So, we store the value 1 in register X11 and subtract X11 from X9 at the end of each loop, counting down until the list is/should be empty. Now, the loop can start. We load the first value (at position 0) from the list, that is in register X7, and store it in register X12. Next, we add that freshly pulled value to register X0. Register X0 will store the current, and at the end, the total sum of the list's values from X7. The next step is to go to the next value of the list. So, we go 8 Bytes forward in the list, by adding X10 (value 8) to the register X7. Now, we count down the loop counter, by subtracting X11 from X9. This way, we will know that there is one value less in the list stored in X7 and therefore, there will be one loop less now conducted. If X9 is 0, the loop will end. If X9 is not empty (B.NE), the loop continues. The previous line of code used SUBS instead of SUB. This is because you want to flag the value to the line below (B.NE). This way, the loop condition can be checked. If each value of the list is successfully added and the loop condition is not fulfilled anymore (so, X9 is equal to 0), the loop stops and X0 shows the full sum of the list's values. The final result is $1 + 4 + 1 + 5 + 9 + 2 = 22$.