**Name:** Avula Puneeth Kumar Reddy

**Rollno:** 2022IMG001

**Objective1:**  Write a program to load the value 0xFFEDFC00 into r0. Try to minimise the number of instructions.

**Methodology:**

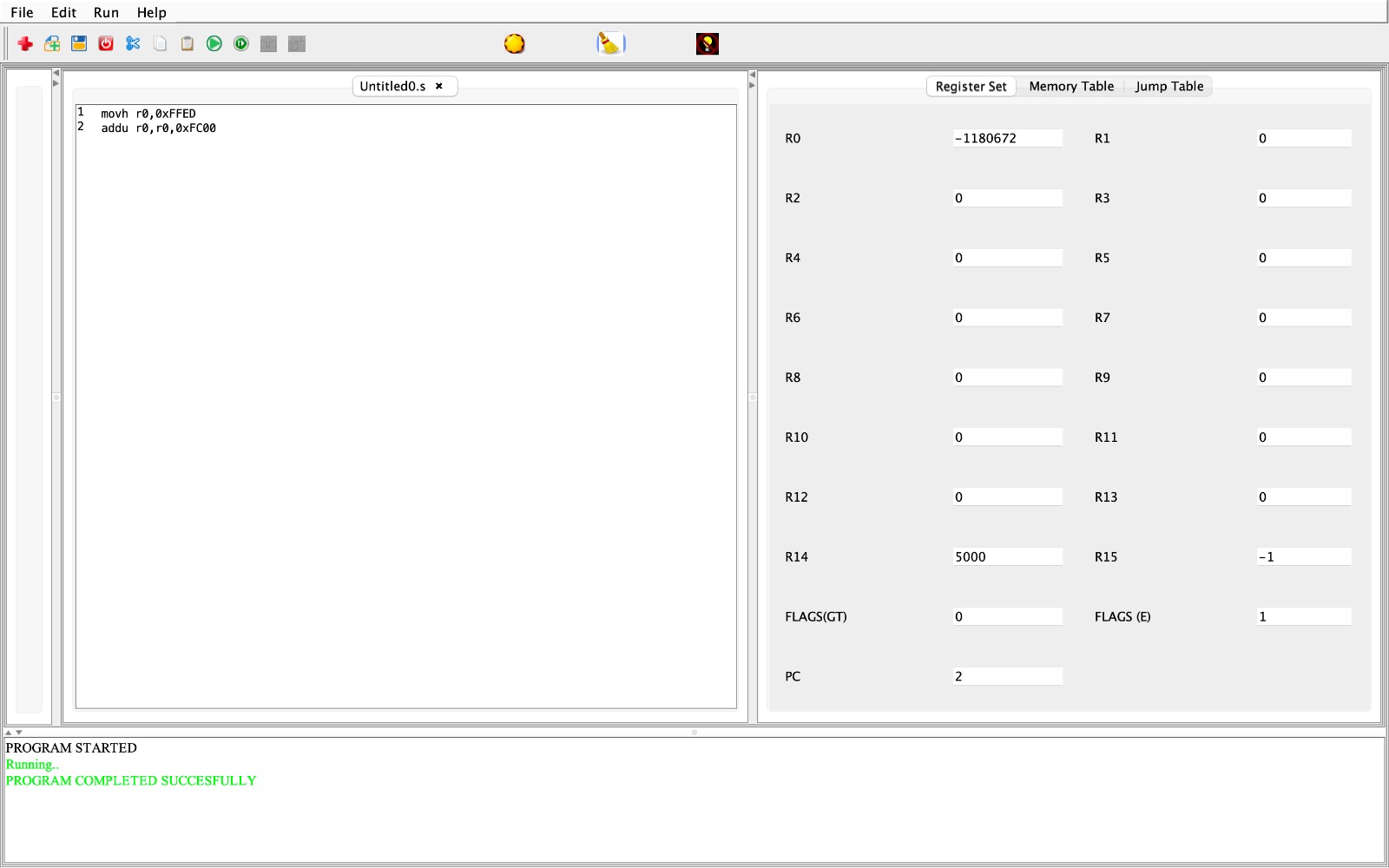
1. The first instruction, movh r0,0xFFED, moves the value 0xFFED to the upper 16 bits of register r0.
2. The second instruction, addu r0,r0,0xFC00, adds the value 0xFC00 to register r0.
3. The final value of register r0 is 0xFFED + 0xFC00 = 0x1C4ED.

**Code:**

movh r0,0xFFED

addu r0,r0,0xFC00

**Output:**



**Objective2:** Write an assembly program to set the 5th bit of register r0 to the value of the 3rd bit of r1. Keep the rest of the contents of r0 the  
same. The convention is that the LSB is the first bit, and the MSB is  
the 32nd bit.  Try to minimise the number of instructions.

**Methodology:**

1. The program starts by moving the value 15 to register r4.
2. It then moves the value 1 to register r5.
3. It then moves the value 0x0000 to the high 16 bits of register r0.
4. It then adds the value 0x0067 to register r0.
5. It then moves the value 0x0000 to the high 16 bits of register r1.
6. It then adds the value 0x000C to register r1.
7. It then shifts register r1 to the right by 2 bits.
8. It then performs a bitwise AND operation on registers r1 and r5 and stores the result in register r2.
9. It then performs a bitwise AND operation on registers r0 and r4 and stores the result in register r3.
10. It then shifts register r0 to the left by 1 bit.
11. It then adds register r2 to register r0.
12. It then shifts register r0 to the left by 4 bits.
13. It then adds register r3 to register r0.

**Code:**

.main:

mov r4,15

mov r5,1

movh r0,0x0000

addu r0,r0,0x0067

movh r1,0x0000

addu r1,r1,0x000C

lsr r1,r1,2

and r2,r1,r5

and r3,r0,r4

lsr r0,r0,5

lsl r0,r0,1

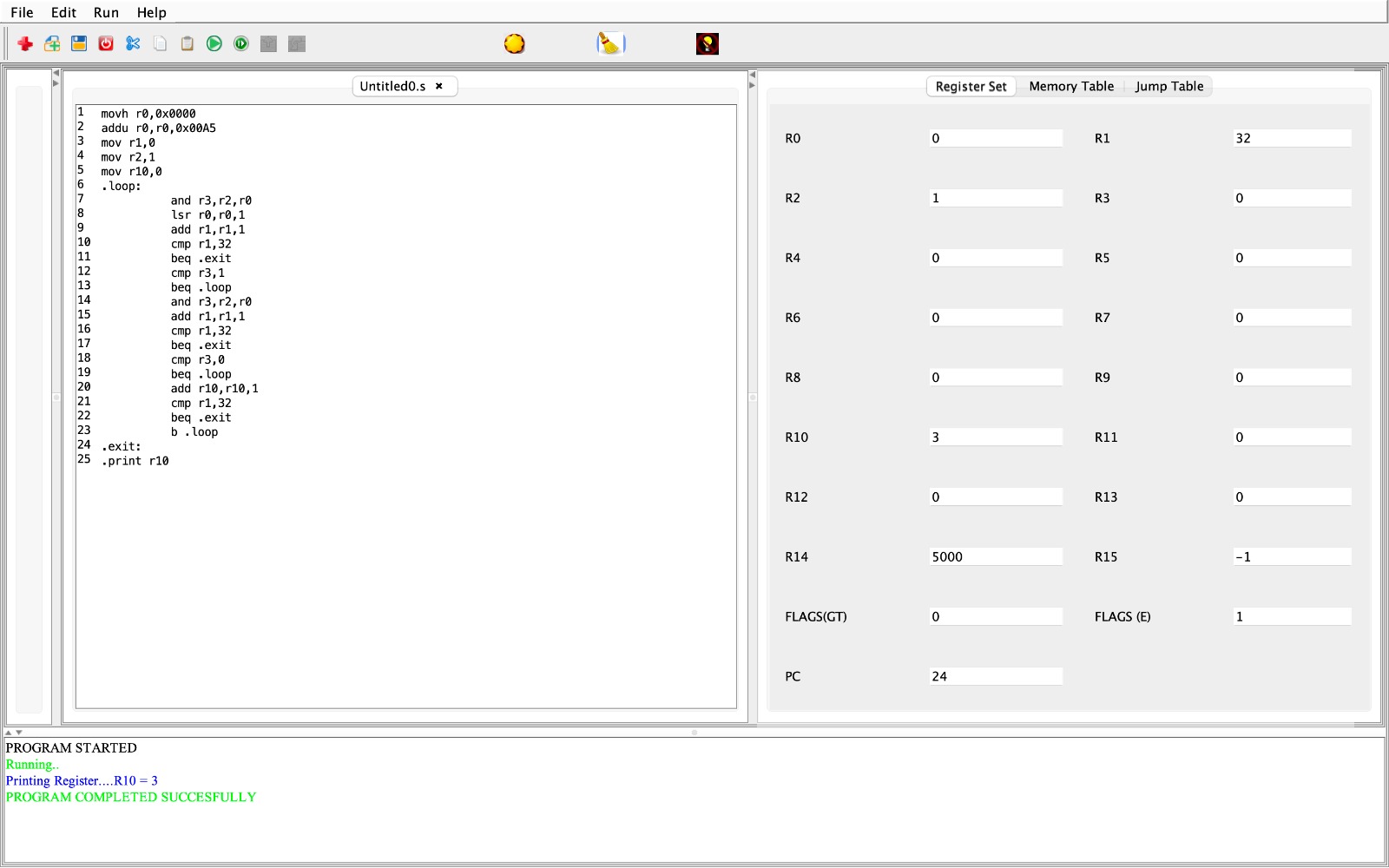
add r0,r0,r2

lsl r0,r0,4

add r0,r0,r3

.exit:

**Output:**



**Objective3:**  Write a program in SimpleRisc assembly to convert an integer stored in memory  
from the little endian to the big endian format.

**Methodology:**

1. It moves the value 0x1234 to the high 16 bits of register r0.
2. It adds the value 0x5678 to register r0.
3. It shifts register r0 to the left by 24 bits and stores the result in register r2.
4. It shifts register r0 to the right by 24 bits and stores the result in register r3.
5. It moves the value 0xFF00 to register r6.
6. It moves the value 0xFF0000 to register r7.
7. It performs a bitwise AND operation on registers r0 and r6 and stores the result in register r4.
8. It shifts register r4 to the left by 8 bits.
9. It performs a bitwise AND operation on registers r0 and r7 and stores the result in register r5.
10. It shifts register r5 to the right by 8 bits.
11. It adds registers r3 and r5 and stores the result in register r3.
12. It adds registers r2 and r4 and stores the result in register r2.
13. It adds registers r2 and r3 and stores the result in register r2.
14. It moves the value of register r2 to register r0.

**Code:**

movh r0,0x1234

addu r0,r0,0x5678

lsl r2,r0,24

lsr r3,r0,24

mov r6,0xFF00

mov r7 0xFF0000

and r4,r0,r6

lsl r4,r4,8

and r5,r0,r7

lsr r5,r5,8

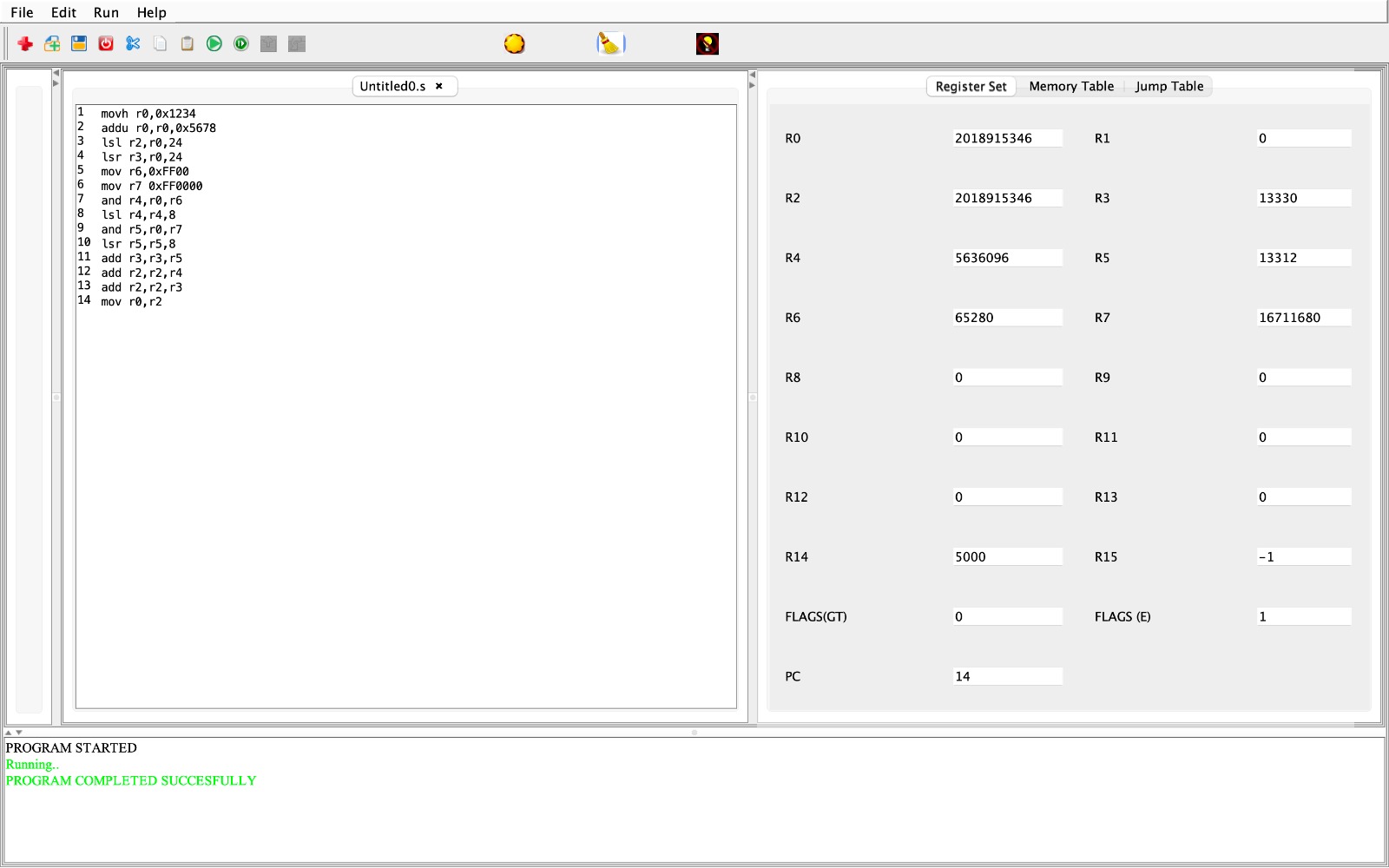
add r3,r3,r5

add r2,r2,r4

add r2,r2,r3

mov r0,r2

**Output:**



**Objective4:** Given a 32-bit integer in r3, write a SimpleRisc assembly program to count the number of 1 to 0 transitions in it.

**Methodology:**

1. Inside the loop, it performs the following steps:
   * It performs a bitwise AND operation on registers r2 and r0 and stores the result in register r3.
   * It then shifts register r0 to the right by one bit.
   * It then adds 1 to register r1.
   * It then compares register r1 to the value 32.
   * If register r1 is equal to 32, the program jumps to the label .exit.
   * It then compares register r3 to the value 1.
   * If register r3 is equal to 1, the program jumps to the label .loop.
   * It then performs a bitwise AND operation on registers r2 and r0 and stores the result in register r3.
   * It then adds 1 to register r1.
   * It then compares register r1 to the value 32.
   * If register r1 is equal to 32, the program jumps to the label .exit.
   * It then compares register r3 to the value 0.
   * If register r3 is equal to 0, the program jumps to the label .loop.
   * It then adds 1 to register r10.
   * It then compares register r1 to the value 32.
   * If register r1 is equal to 32, the program jumps to the label .exit.
   * Otherwise, the program jumps back to the label .loop.
2. When the program reaches the label .exit, it prints the value of register r10 to the console.

**Code:**

.main:

movh r0,0x0000

addu r0,r0,0x00A5

mov r1,0

mov r2,1

mov r10,0

.loop:

and r3,r2,r0

lsr r0,r0,1

add r1,r1,1

cmp r1,32

beq .exit

cmp r3,1

beq .loop

and r3,r2,r0

add r1,r1,1

cmp r1,32

beq .exit

cmp r3,0

beq .loop

add r10,r10,1

cmp r1,32

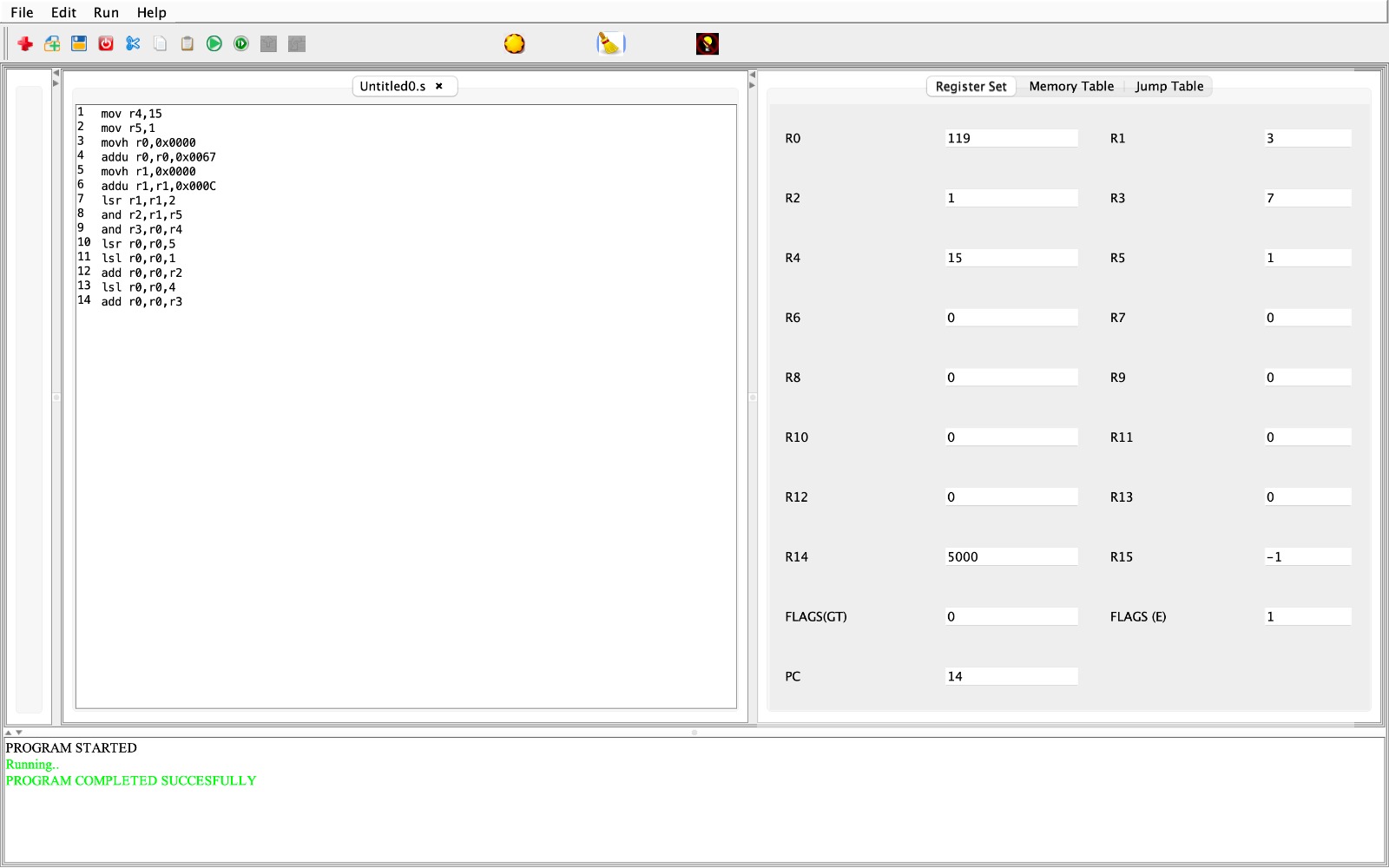
beq .exit

b .loop

.exit:

.print r10

**Output:**



**Objective5:** Design a SimpleRisc program that examines a 32-bit value stored in r1 and counts the number of contiguous sequences of 1s. For example, the value: 01110001000111101100011100011111 contains six sequences of 1s. Write the result in r2.

**Methodology:**

1. The program starts by moving the value 0x711E to register r0.
2. It then adds the value 0xC71F to register r0.
3. It then moves the value 0 to register r1.
4. It then moves the value 1 to register r2.
5. It then moves the value 0 to register r10.
6. It then enters a loop labeled .loop.
7. Inside the loop, it performs the following steps:
   * It performs a bitwise AND operation on registers r2 and r0 and stores the result in register r3.
   * It then shifts register r0 to the right by one bit.
   * It then adds 1 to register r1.
   * It then compares register r1 to the value 32.
   * If register r1 is equal to 32, the program jumps to the label .exit.
   * It then compares register r3 to the value 0.
   * If register r3 is equal to 0, the program jumps to the label .loop.
   * Otherwise, it adds 1 to register r10.
8. After the loop has finished executing, the program enters a second loop labeled .loop1.
9. Inside the loop1 loop, the program performs the following steps:
   * It performs a bitwise AND operation on registers r2 and r0 and stores the result in register r3.
   * It then shifts register r0 to the right by one bit.
   * It then adds 1 to register r1.
   * It then compares register r1 to the value 32.
   * If register r1 is equal to 32, the program jumps to the label .exit.
   * If register r3 is equal to 0, the program jumps to the label .loop.
   * Otherwise, the program compares register r1 to the value 32.
   * If register r1 is equal to 32, the program jumps to the label .exit.
   * Otherwise, the program jumps back to the label .loop1.
10. When the program reaches the label .exit, it prints the value of register r10 to the console.

**Code:**

.main:

movh r0,0x711E

addu r0,r0,0xC71F

mov r1,0

mov r2,1

mov r10,0

.loop:

and r3,r2,r0

lsr r0,r0,1

add r1,r1,1

cmp r1,32

beq .exit

cmp r3,0

beq .loop

add r10,r10,1

.loop1:

and r3,r2,r0

lsr r0,r0,1

add r1,r1,1

cmp r1,32

beq .exit

cmp r3,0

beq .loop

cmp r1,32

beq .exit

b .loop1

.exit:

.print r10

**Output:**

