Name: Abu Butt

Professor: Izidor Gertner

Class: CSC 43200

Date: July 8, 2017

Topic: Report on ARM Emulator

Introduction:

ARM stands for originally Acorn RISC Machine, later Advanced RISC Machine, is a family of reduced instruction set computing architectures for computer processors, configured for various environments. Main features of the ARM Instruction Set are all instructions are 32 bits long. Most instructions are executed in a single cycle. Most instructions can be conditionally executed. It is a load/store architecture. It has 3 operands format. In ARM instruction sets there are 16 registers and each register is a 32-bit number. Registers are used to store data.

ARM Emulator (VisUAL) Examples:

Adding Numbers:

Code in C:

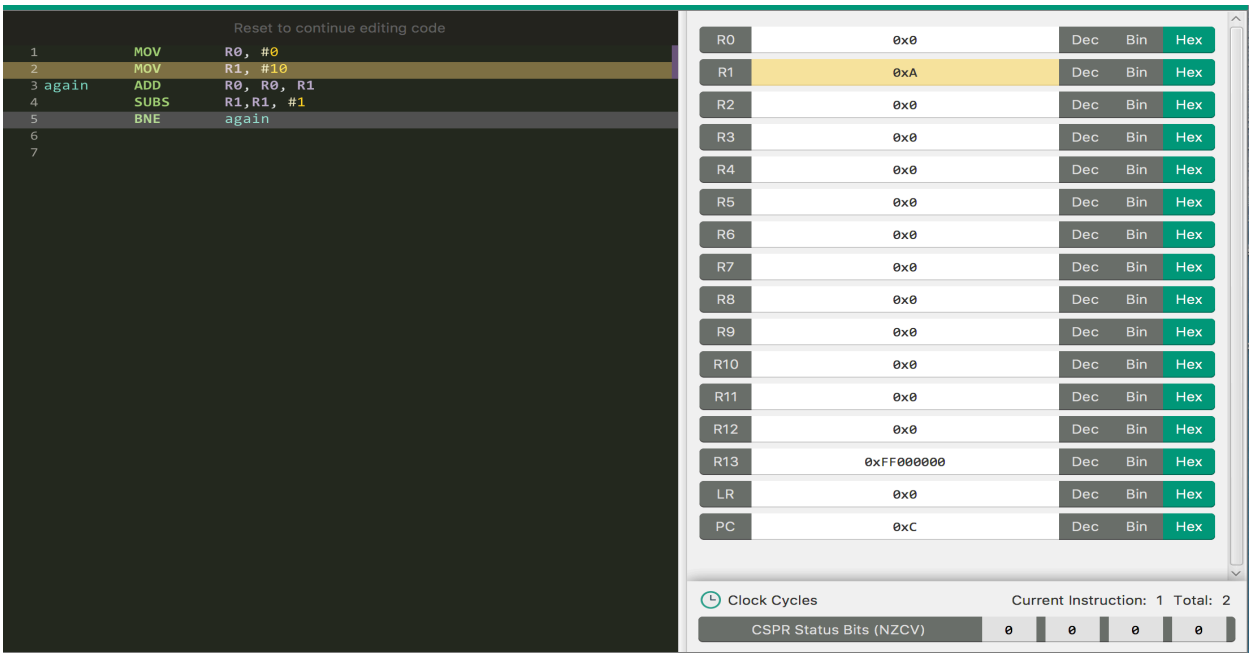
```
int total;
    int i;

    total = 0;
    for (i = 10; i > 0; i--) {
        total += i;
    }
```

Above is the code for adding numbers from 1 to 10. This code is written in C. The sum of all numbers from 1 to 10 is 55 in decimal.

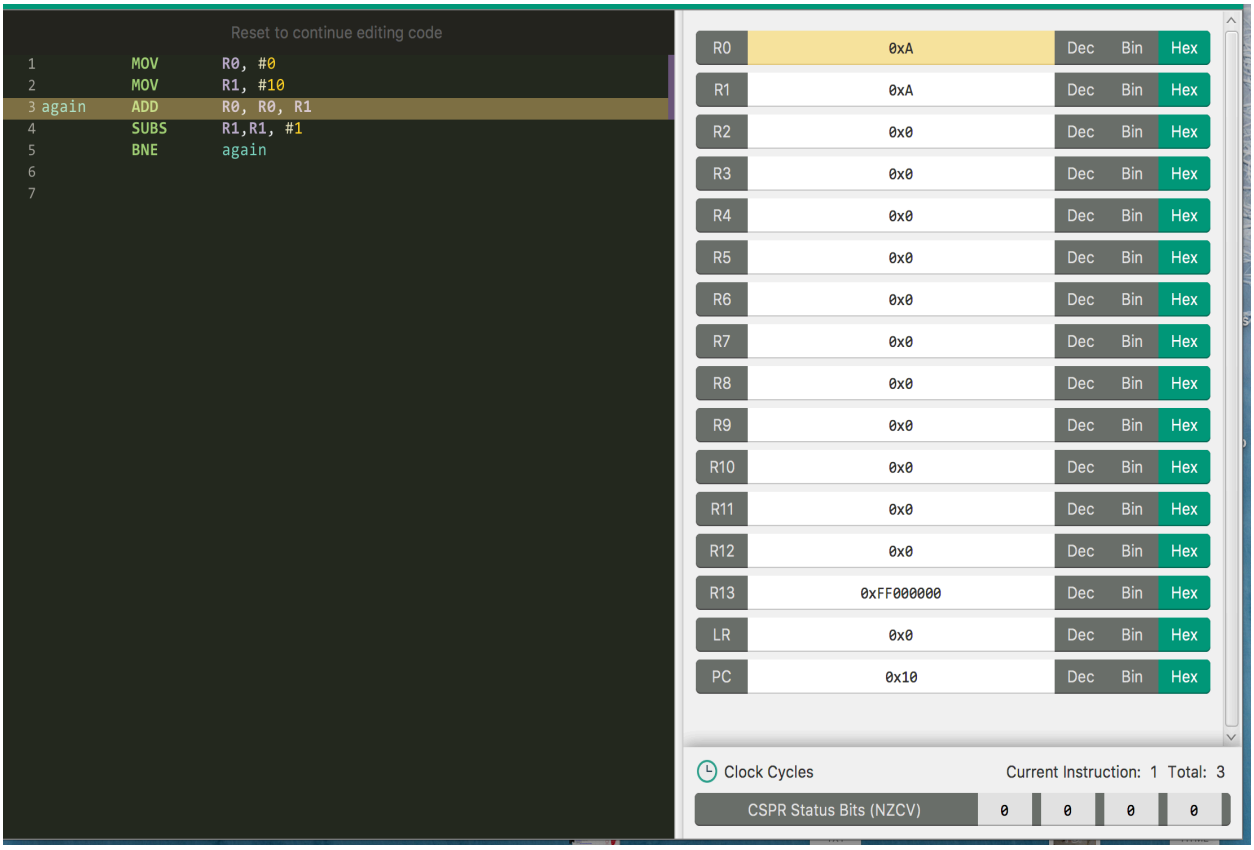
ARM ISA:

Example 1: Adding Numbers

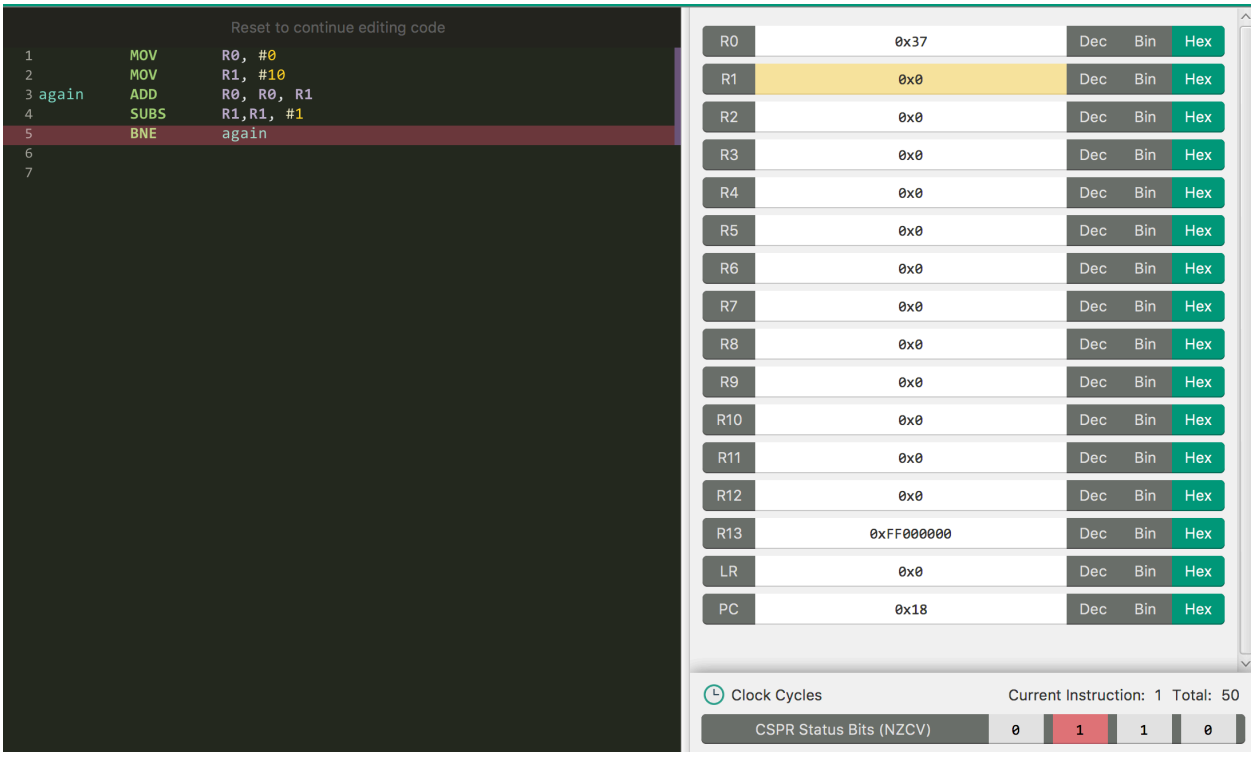


This is the ARM ISA. On the left side of the screen are the ARM instructions for adding numbers from 1 to 10. The right side has all the register that are being used in this problem. This is the beginning of the code. So, you can see the registers R0 and R1 are initialized with their values. The values are in hexadecimal. It also has register PC, which is a program counter,

which is responsible for keeping the instructions that will be executed next.



This picture is for the loop that is executed in the code. As you can see the again statement in the ARM instructions which mean keep executing the statements until the both values are equal. This loop will stop executing when the branches (values) are equal. In the loop, we are adding the values of registers R0 and R, and storing it in the register R0. Then we decrement the value of register R1 by 1. This execution can be seen in the right side of the emulator.



This is the end of the execution, where register R0 has the value of 37 on hexadecimal, which in decimal is 55. Register R1 is 0. The execution stopped when both branches (values) were equal.

Example 2: Hailstone Sequence3

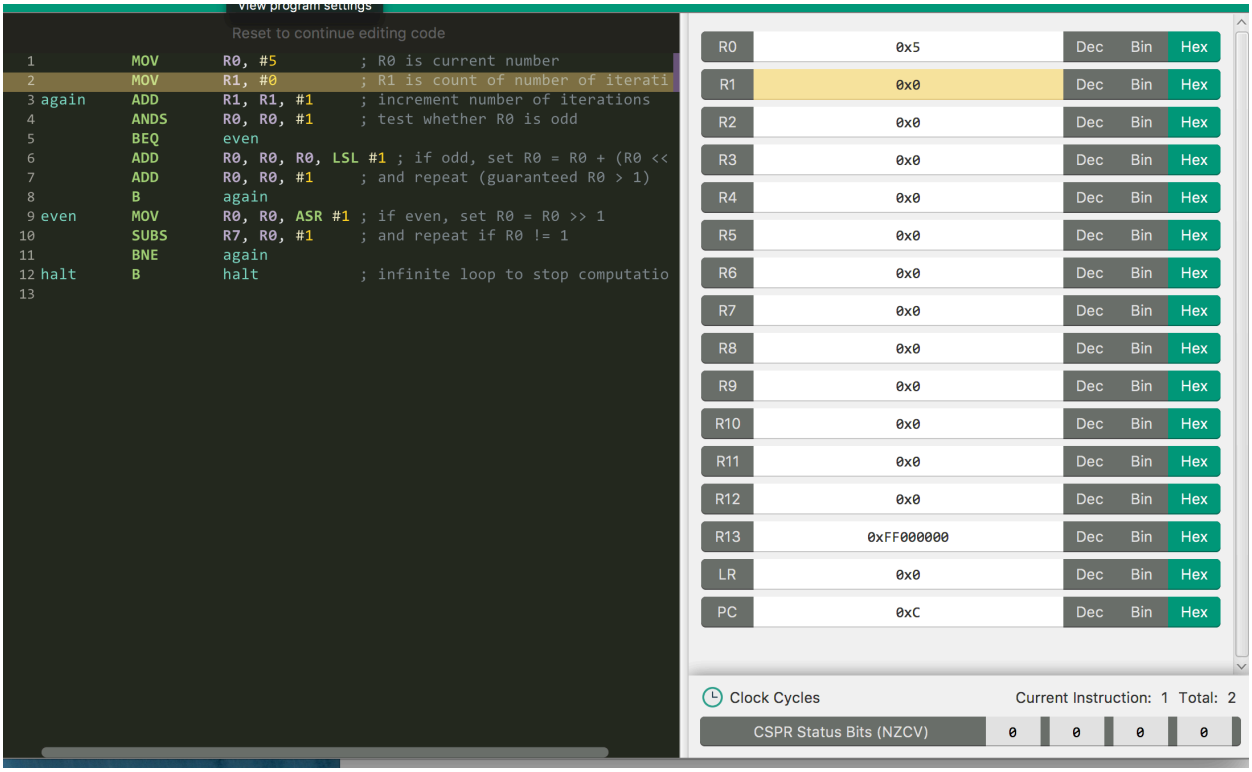
iters ← 0
while $n \neq 1$**:**
  iters ← *iters* + 1
  if n is odd**:**
    $n \leftarrow 3 \cdot n + 1$
  else:
    $n \leftarrow n / 2$

This is the code in C. It means we are given an integer n, we repeatedly apply the top procedure.

If n is odd we execute this line of code $n \leftarrow 3 \cdot n + 1$, otherwise $n \leftarrow n / 2$.

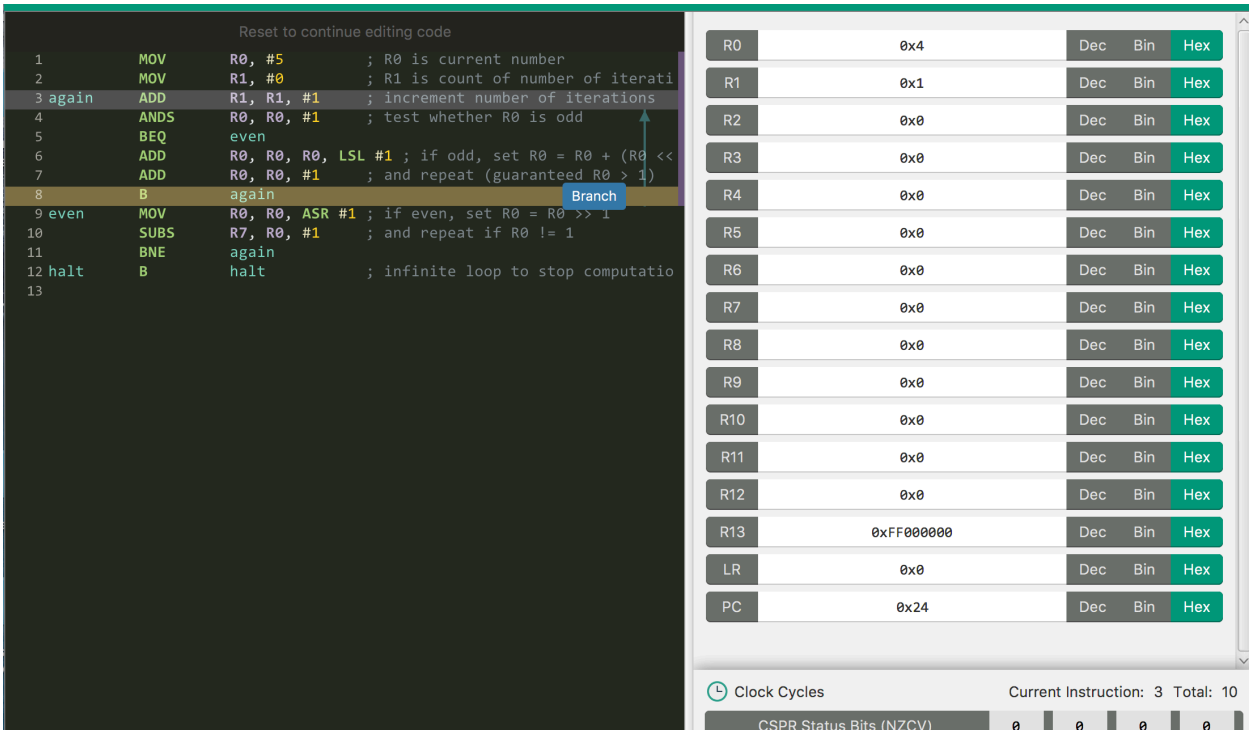
ARM ISA:

Example 2: Hailstone Sequence

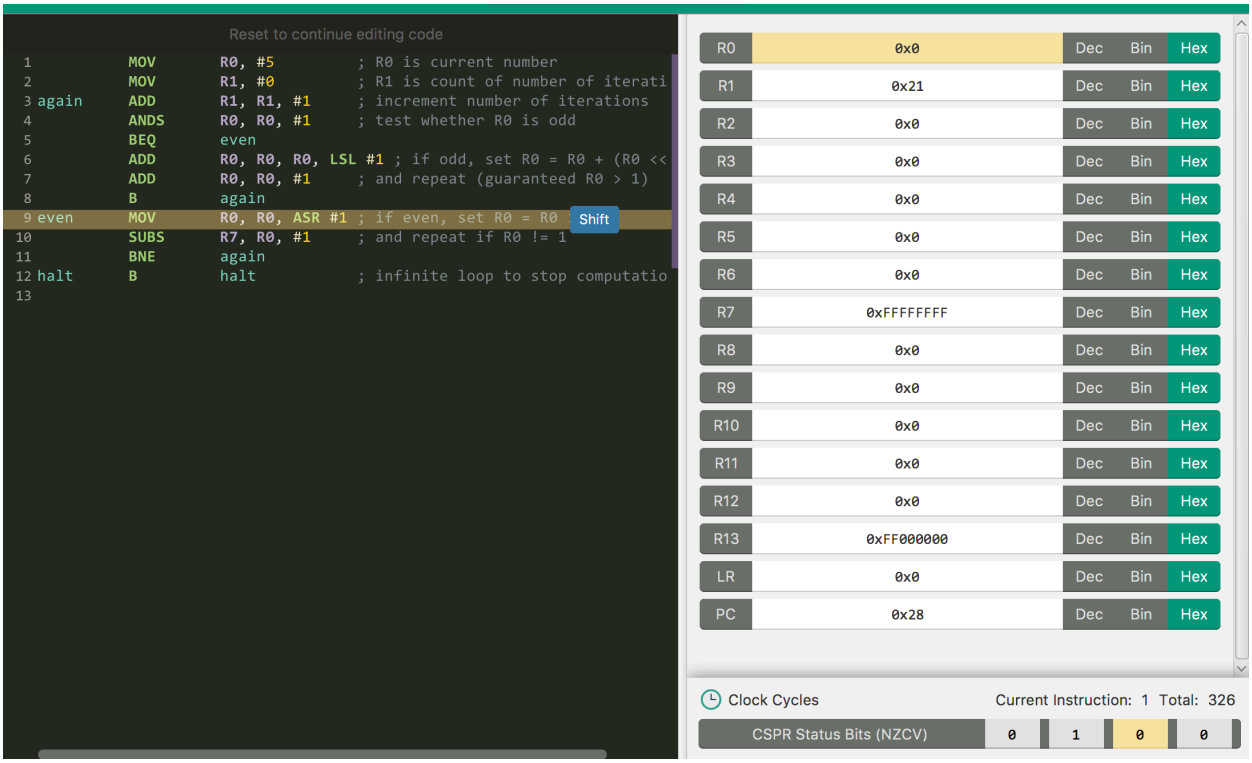


This is the ARM ISA. On the left side of the screen are the ARM instructions for Hailstone Sequence. The right side has all the register that are being used in this problem. This is the beginning of the code. So, you can see the registers R0 and R1 are initialized with their values. The values are in hexadecimal. It also has register PC, which is a program counter, which is

responsible for keeping the instructions that will be executed next.



Now from lines 3 to line 11 is the while loop and the if else condition. First it increments the numbers of iterations. Then it tests if the value of register R0 is even or odd. If it is odd, then lines 6 and 7 are executed else lines 9 and 10 are executed. The changes in all the register can also be seen on the right side.



This piece of code will keep running because we do not have the terminate condition. It will keep running either the value is odd or even.

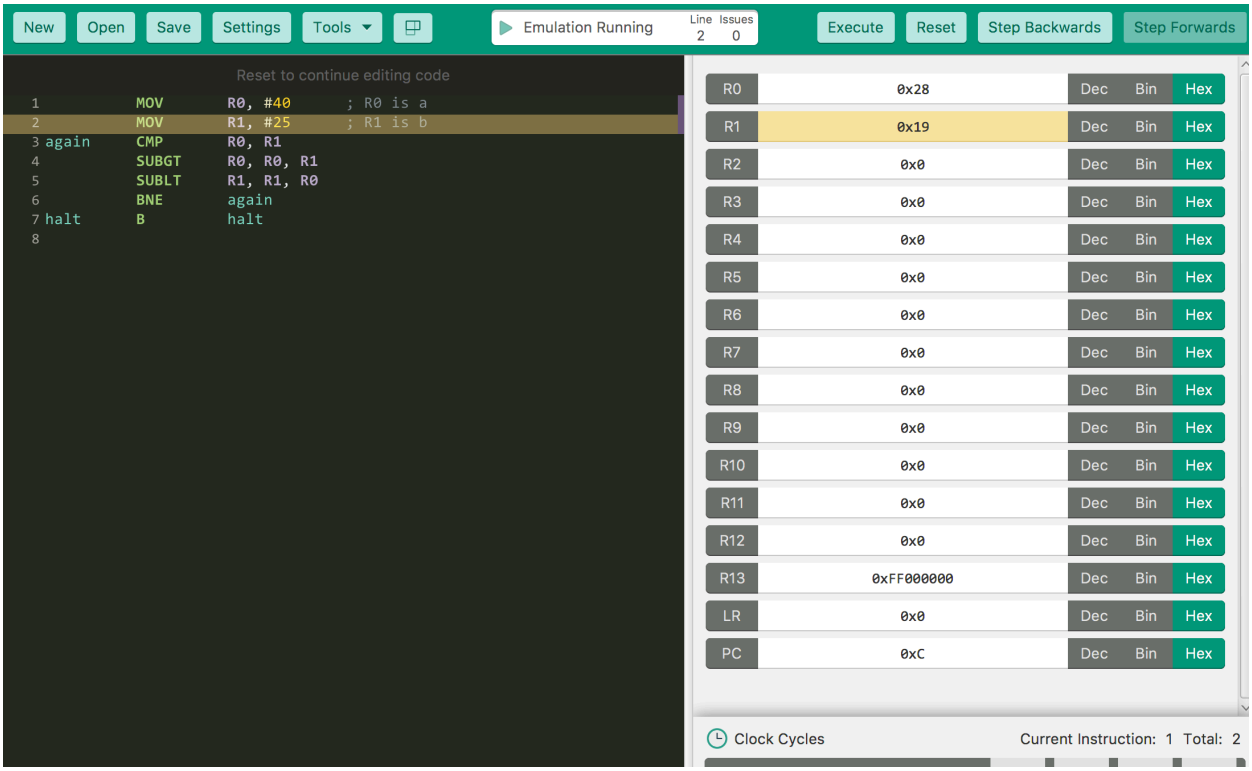
Example 3: Condition Codes

```
a = 40;
    b = 25;
    while (a != b) {
        if (a > b) a -= b;
        else       b -= a;
    }
```

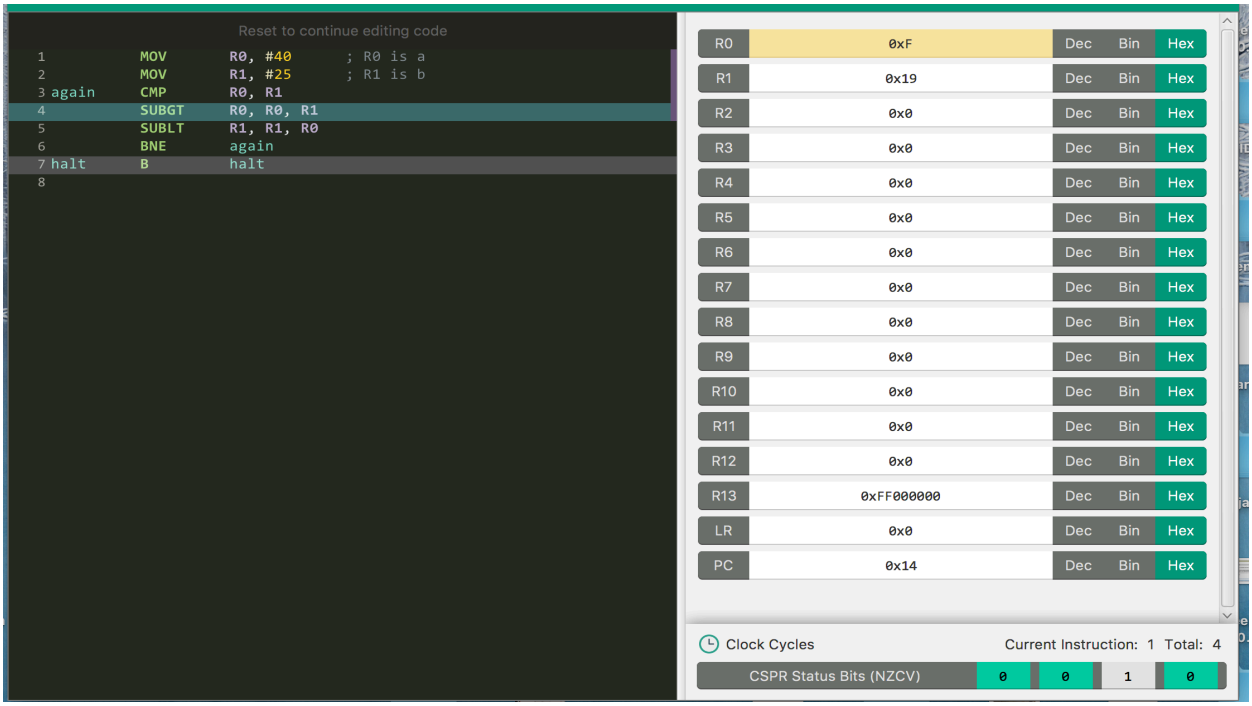
This is the code in C. It means that the operation should take place only when certain combinations of the combinations of the flags hold. You can specify the condition code by including it as part of the opcode.

ARM ISA:

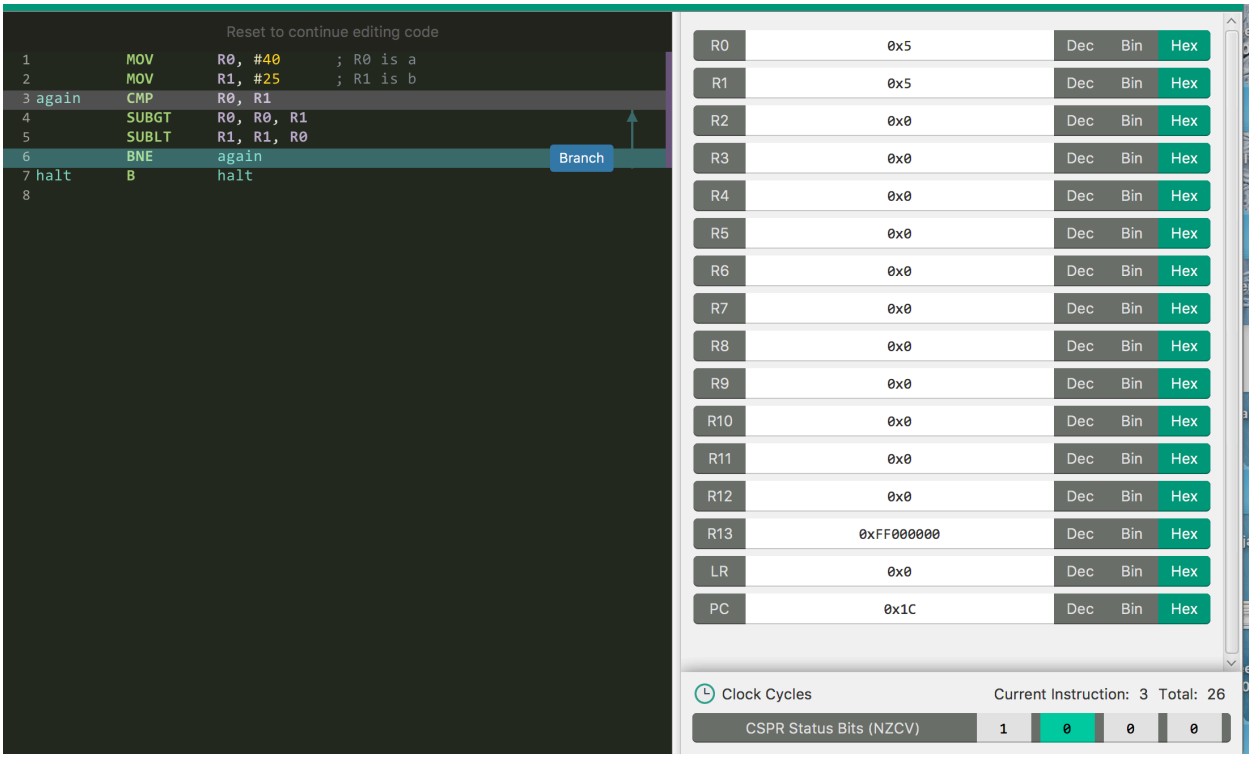
Example 3: Condition Codes



This is the ARM ISA. On the left side of the screen are the ARM instructions for Condition Codes. The right side has all the register that are being used in this problem. This is the beginning of the code. So, you can see the registers R0 and R1 are initialized with their values. The values are in hexadecimal. It also has register PC, which is a program counter, which is responsible for keeping the instructions that will be executed next.



Line 3 of the code compares the value of the registers R0 and R1, stores it in register R0. Then line 4 and 5 checks which values is greater and lesser, then it subtracts the values from the registers.



At the end, the value of register R0, R1 have the same value of 5.