Lab1 Report

name : Zhilong Chen   student ID : 3200103390

Lab Name and Lab task

Lucky 111: Professor Patt loves the number 7. As a computer man he would represent 7 in binary 111. That is Patt's favorite binary pattern, called Lucky 111. Your job is to write a program to judge whether a 16-bit value contains that pattern (three consecutive 1's).

Task:

- You are required to write in LC-3 machine codes (0's and 1's).
- Your program should start at x3000, which means the first instruction of your program is located in position x3000. The input 16-bit value is located in memory location x3100.
- Your program should load the value and then exanimate it. If the input value satisfies, then set R2 to 1. Otherwise, set R2 to 0.
- Your program must halt after examining the value.

Algorithm

A simple idea of this problem is that if we want to know whether a 3-bit value x contains three consecutive 1's, we can carry out the option $x \wedge (111)_2$, if the answer is equal to $(111)_2$, then we can know x has three consecutive 1's in bit $[0:2]$.

Then if we use $(1110)_2$, $(11100)_2$, and so on, to replace $(111)_2$, then we can check whether the 16-bit value x has three consecutive 1's at $[1:3]$, $[2:4]$ and so on. Using this algorithm, we can use at most 13 steps to check whether x has three consecutive 1's.

Core code

```
;; at first R1 is assigned with 7 and R3 is assigned with 13
0101 010 000 000 001 ;; R2 <- R0 and R1
1001 010 010 111111
0001 010 010 1 00001 ;; R2 <- -R2
0001 010 010 000 001 ;; R2 <- R2 + R1

0000 010 000000100 ;; if R2 = 0, then goto the end and R2 <- 1
0001 001 001 000 001 ;; R1 <- R1 * 2
0001 011 011 1 11111 ;; R3 <- R3 - 1
0000 010 000000100 ;; if R3 = 0, then goto the end
0000 111 111110111 ;; goto loop head
```

This code shows the core of this algorithm.

Meaning of every register

R_0: load the 16-bit x stored at x3100.

R_1: the check number with three consecutive 1's. Simply, it is $(111)_2$, $(1110)_2$, $(11100)_2$ and so on.

R_2: store the result $R_0 \wedge R_1$, and know whether x contains three consecutive 1's in bit $[i : i+2]$ by judging whether $-R_2 + R1$ is zero.

R_3: the count number with the initial value of 13, which means this check algorithm will be carried out at most 13 steps.

Implementation of the algorithm

We use one instruction of NOT and ADD to implement the option that change R_2 to $-R_2$, and we can know whether x contains three consecutive 1's in bit $[13 - R_3 : 15 - R_3]$ by judging whether $-R_2 + R1$ is zero. Then let $R_2 = R_1 - R_2$, if R_2 equals to 0, which means x contains three consecutive 1's, then we can go to the end and let $R_2 = 1$, showing the final answer.

After the check, if R_2 is not zero, then we let $R_3 = R_3 - 1$ and $R_1 = R_1 \times 2$ and go to the loop head to continue the loop. Once the R_3 becomes zero, which means we have check the bit $[13 : 15]$ and x doesn't contain three consecutive 1's, then we can go to the end and let $R_2 = 0$, showing the final answer.

Improvement according to the TA's question

In the algorithm, we use R_3 as a counter. When R_3 become zero, then we stop the loop. But noted that when R_3 become zero, R_1 is $(1110000000000000)_2$, which is a negative number and when R_3 is not zero, R_1 is a positive number. So we can use R_1 as a condition to judge the loop end. In this way can we use one lesser register.