中国科学技术大学计算机学院
《计算机系统概论》实验报告

LAB 01: A LC-3 Machine Language Program

姓名:王晨 学号:PE20060014

完成日期:2020 年 11 月 18 日

一、实验要求：

Write a program in LC-3 machine language that rotates a given value of 16-bits by 2 bits to the left.

1. The initial bit pattern is in memory location x3100

2. The rotate amount is stored in memory location x3101.

3. Using those values, your program should perform the left rotation and store the result in memory location x3102.

4. Your program should start at memory location x3000.

二、算法思路：

对于将二进制串左移的方法，可以参照x86汇编的SHL指令作修改，首先用一个移位寄存器(不妨设为R3)存储该二进制串的最高位(保存自相加后可能溢出的位)。然后将二进制串自相加1次，这样得到的二进制串会整体左移1位，末位补0。此时如果R3=1，则原二进制串首位为1，将移位后的二进制串再加上1即可；如果R3=0，则不需要加1。因此，最后完成翻转后的二进制串是原二进制串自相加后，再加上移位寄存器R3的值。

下面用一表格作辅助说明，以4位二进制串1010为例，存放在R2：

步骤	执行操作前的R2	执行操作前的R3	执行操作	执行操作后的R2	执行操作后的R3
1	1010	0	R3保存R2首位	1010	1
2	1010	1	R2 <- R2+R2	0100	1
3	0100	1	R2 <- R2+R3	0101	1

那么如果要左移翻转2bit位，只需将上述操作循环2次，左移翻转n位，循环n次即可。

下面用一流程图作辅助说明，左移翻转位数N存放在R1，二进制串存放在R2，R3为移位寄存器，存放每次操作前R2的首位：

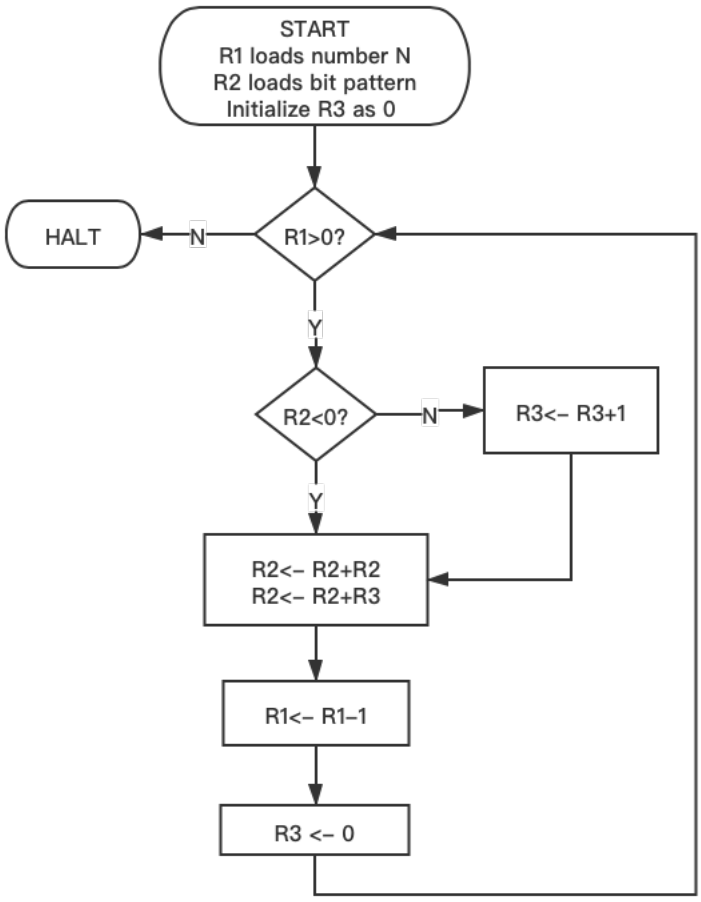


图1:算法流程图

从算法流程图1中可以看到这里一共对R1，R2各做了一次符号判断，可以通过BR指令实现此不同符号结果下的指令跳转。

三、程序代码及注释：

LC3Tools

bitsRotation.bin

```
0011 000000000000   ; start the program at location x3000
1110 000 011111111  ; R0 <-PC+0x0FF ,makes R0 <- x3100
0101 011 011 1 00000 ; R3 <- 0
0110 001 000 000001  ; R1 <- MEM[R0+1] R1 Load 需要移位的次数
0110 010 000 000000  ; R2 <- MEM[R0] Load 待翻转的二进制串
0101 010 010 1 11111 ; R2=R2, set NZP, Here is the loop entry.
0000 011 000000001   ; 判断R2首位是否为0,是则JMP to line 9指令,否则R3=1
0001 011 011 1 00001 ; R3 <- 1
0001 010 010 000 010 ; R2 <- R2+R2
0001 010 010 000 011 ; R2 <- R2+R3
0101 011 011 1 00000 ; R3 <- 0
0001 001 001 1 11111 ; R1 <- R1-1
0000 101 111111000   ; 判断R1是否为0,是则保存,否则JMP to line 6
0111 010 000 000010  ; MEM[x3102] <- R2 , Save result to MEM[x3102]
1111 0000 00100101   ; HALT
```

① 0011 000000000000 ; start the program at location x3000

② 1110 000 011111111 ; LEA R0,#0x0FF, makes R0 <- x3100

③ 0110 001 000 000001 ; R1 <- MEM[R0+1], R1 Loads the counter number

④ 0110 010 000 000000 ; R2 <- MEM[R0] , R2 Loads the initial bit pattern

⑤ 0101 011 011 1 00000 ; AND R3,R3,00000 ; R3 <- 0

⑥ 0101 010 010 1 11111 ; AND R2,R2,#11111, setCC, Here also is the loop entry.

⑦ 0000 011 000000001 ; 判断R2首位是否为0,是则BR to line 9指令

⑧ 0001 011 011 1 00001 ; ADD R3 #1

⑨ 0001 010 010 000 010 ; ADD R2,R2,R2; 即R2 <- R2+R2

⑩ 0001 010 010 000 011 ; ADD R2,R2,R3; 即R2 <- R2+R3

⑪ 0101 011 011 1 00000 ; R3 <- 0; 使R3重置为0

⑫ 0001 001 001 1 11111 ; ADD R1,R1,#-1; 即R1 <- R1-1,

⑬ 0000 101 111111000 ; 判断R1是否为0,是则执行STR,否则BR to line 6

⑭ 0111 010 000 000010 ; STR R2,R0,x000002, Save result to MEM[x3102]

⑮ 1111 0000 00100101 ; HALT

解释:

1. Line2 用了LEA指令，让R0保存x3100，以便后续访存x3100～x3102内的数据。
2. Line3,4 是让R1到x3100装载左移反转的位数N，R2到x3101装载initial bit pattern
3. Line6,7是关键的一步，让R2和#11111相与，R2本身的值是不发生变化的，但是可以使得PSR的CC产生标识，然后BR指令根据其CC和NZP的情况作指令跳转，由于每次循环都需要判断R2的首位是1还是0作后续操作，因此这步可以作为循环的入口。当然，循环入口也可以设为判断R1或者R3置0这步，只是在代码顺序上有调整，没有本质区别。这里因为R1是自己输入的值，不会是0和负数，因此，我将R1的正负判断放在了最后，将循环入口设为了Line6。
4. Line10之后已经完成了1bit位的左移翻转，Line13判断R1计数器是否为0，为0则循环结束保存数据到x3102，否则跳转到line6的循环入口。

四、测试用例和结果分析:

这里的测试用例先使用了实验文档中的Example：1101000100001000=xD108存入到memory x3100的位置，计数器的初始值N=2=000000000000010存入到memory x3101的位置，实验结果存入到到memory x3102的位置。

① xD108左移翻转1位应该得到1010001000010001=xA211

② 左移翻转2位应该得到0100010000100011=x4423

③ 左移翻转3位应该得到1000100001000110=x8846

④ 左移翻转16位应该仍然得到xD108

⑤ xFFFF和x0000左移翻转任意位均为其本身

1.将原始数据装载到内存：

Memory

x3100	xD108	-12024	
x3101	x0002	2	
x3102	x0000	0	

图2:实验开始前将原始二进制串和计数器N的值分别存入x3100,x3102

2.单步执行到Line9的时候可以看到下图3，R2为左移1位，末位补0后的值xA210,R3寄存器内的值为1,此时只需将R3加到R2上就完成了1bit位的左移翻转了。

Registers

R0	x3100	12544
R1	x0002	2
R2	xA210	-24048
R3	x0001	1
R4	x0000	0
R5	x0000	0
R6	x0000	0
R7	x0000	0

Memory

x3000	xE0FF	-7937	1110000011111111
x3001	x56E0	22240	0101011011100000
x3002	x6201	25089	0110001000000001
x3003	x6400	25600	0110010000000000
x3004	x54BF	21695	0101010010111111
x3005	x0601	1537	0000011000000001
x3006	x16E1	5857	0001011011100001
x3007	x1482	5250	0001010010000010

图3:单步执行到Line9时的寄存器情况

3.单步执行到Line10的时候可以看到下图4，R2得到xA211,此时说明R3已经加到R2上，完成了1bit位的左移翻转。

Registers

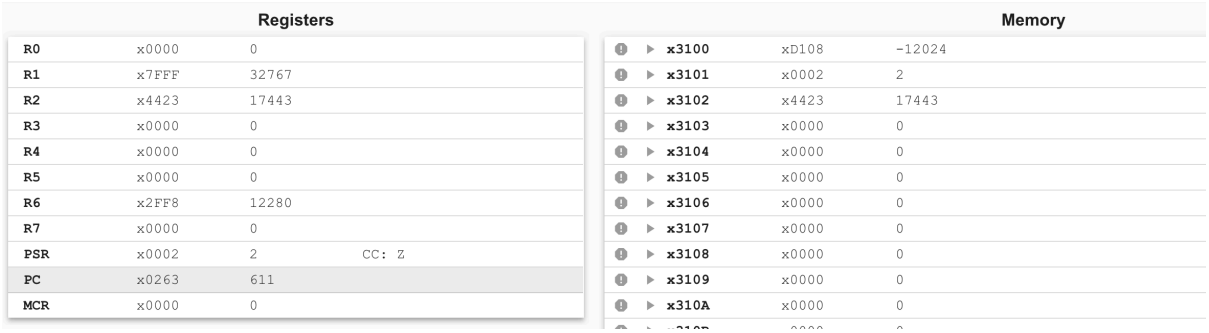
R0	x3100	12544	
R1	x0002	2	
R2	xA211	-24047	
R3	x0001	1	
R4	x0000	0	
R5	x0000	0	
R6	x0000	0	
R7	x0000	0	
PSR	x8004	-32764	CC: N
PC	x3009	12297	
MCR	x0000	0	

Memory

x3000	xE0FF	-7937	1110000011111111
x3001	x56E0	22240	0101011011100000
x3002	x6201	25089	0110001000000001
x3003	x6400	25600	0110010000000000
x3004	x54BF	21695	0101010010111111
x3005	x0601	1537	0000011000000001
x3006	x16E1	5857	0001011011100001
x3007	x1482	5250	0001010010000010
x3008	x1483	5251	0001010010000011
x3009	x56E0	22240	0101011011100000
x300A	x127F	4735	0001001001111111

图4:单步执行到Line10时的寄存器情况

4.全部执行完后可以看到图5，当N=2时，xD108完成2bit位的左移翻转得到x4423，已经存入MEM[x3102]，结果正确：



Registers

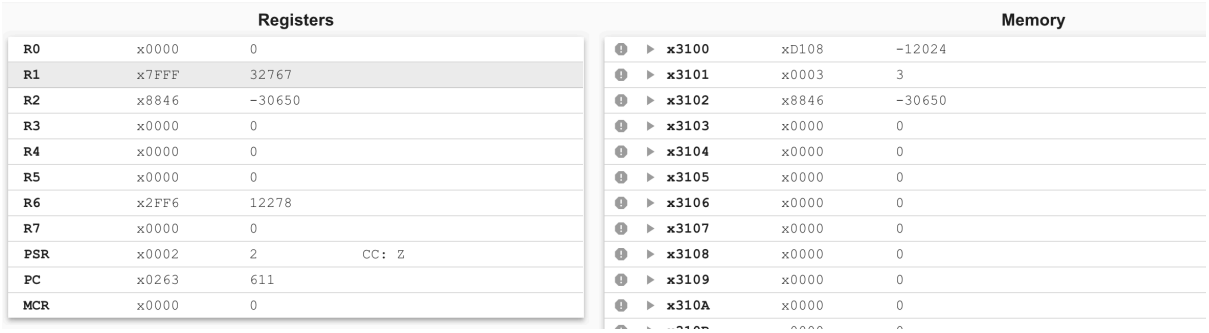
R0	x0000	0	
R1	x7FFF	32767	
R2	x4423	17443	
R3	x0000	0	
R4	x0000	0	
R5	x0000	0	
R6	x2FF8	12280	
R7	x0000	0	
PSR	x0002	2	CC: Z
PC	x0263	611	
MCR	x0000	0	

Memory

x3100	xD108	-12024
x3101	x0002	2
x3102	x4423	17443
x3103	x0000	0
x3104	x0000	0
x3105	x0000	0
x3106	x0000	0
x3107	x0000	0
x3108	x0000	0
x3109	x0000	0
x310A	x0000	0

图5:N=2时执行完后的结果

5.将计数器N存为3进行测试，此时第3次循环时R2的首位为0，最后得到结果x8846，也是正确的：



Registers

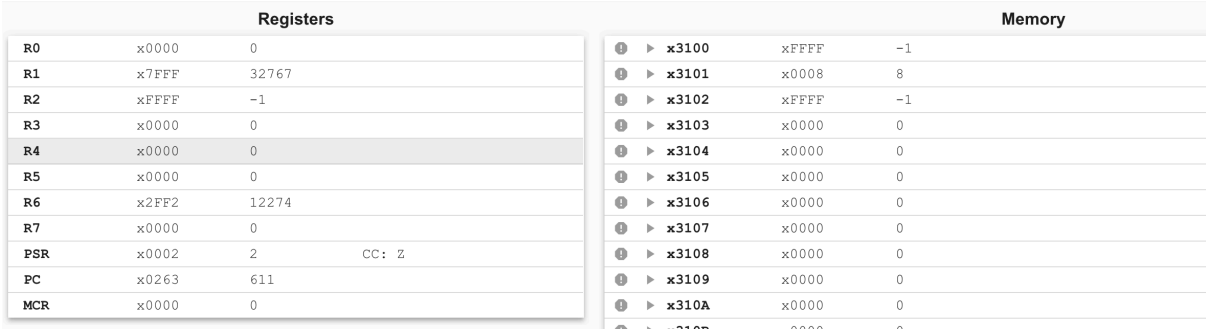
R0	x0000	0	
R1	x7FFF	32767	
R2	x8846	-30650	
R3	x0000	0	
R4	x0000	0	
R5	x0000	0	
R6	x2FF6	12278	
R7	x0000	0	
PSR	x0002	2	CC: Z
PC	x0263	611	
MCR	x0000	0	

Memory

x3100	xD108	-12024
x3101	x0003	3
x3102	x8846	-30650
x3103	x0000	0
x3104	x0000	0
x3105	x0000	0
x3106	x0000	0
x3107	x0000	0
x3108	x0000	0
x3109	x0000	0
x310A	x0000	0

图6:N=3时执行完后的结果

6.将计数器N存为16进行测试，得到结果xD108，也是正确的：



Registers

R0	x0000	0	
R1	x7FFF	32767	
R2	xFFFF	-1	
R3	x0000	0	
R4	x0000	0	
R5	x0000	0	
R6	x2FF2	12274	
R7	x0000	0	
PSR	x0002	2	CC: Z
PC	x0263	611	
MCR	x0000	0	

Memory

x3100	xFFFF	-1
x3101	x0008	8
x3102	xFFFF	-1
x3103	x0000	0
x3104	x0000	0
x3105	x0000	0
x3106	x0000	0
x3107	x0000	0
x3108	x0000	0
x3109	x0000	0
x310A	x0000	0

图7:N=16时执行完后的结果

7.将initial bit pattern存为全1或全0的二进制串进行测试，N=任意数，结果也是正确的：

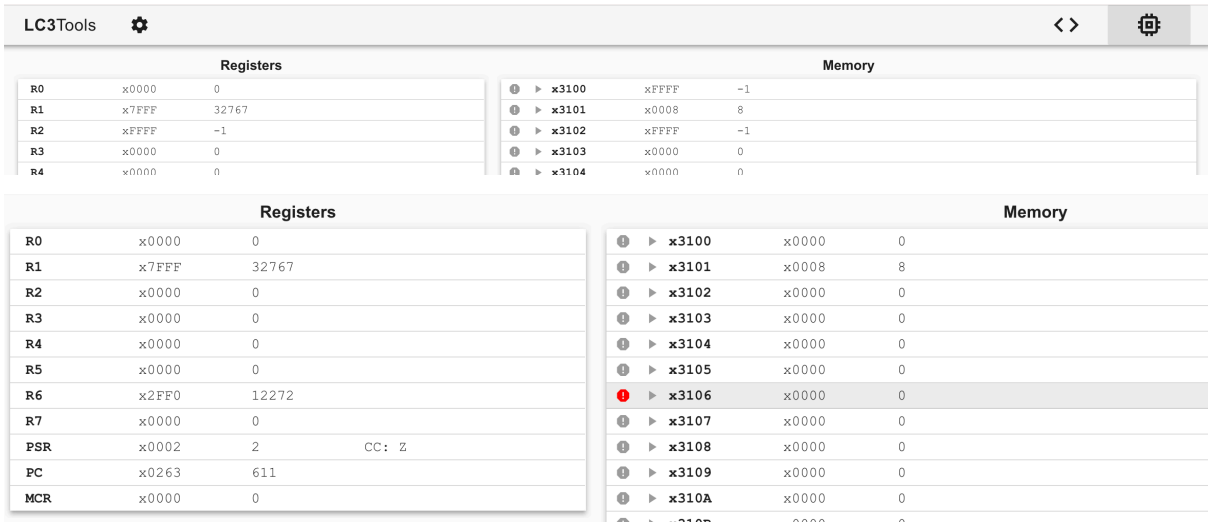


图8:N为任意值时全1串和全0串执行完后的结果

五、实验心得：

1. 本次实验相对比较简单，通过本次实验熟悉了LC3模拟器的使用和调试方法，也大大熟悉了LC3的指令集。

2. 实验难点就在于如何实现一次二进制串的左移，这里想到x86汇编的SHL指令，乘以它的Base2即可，溢出的首位用移位寄存器CF保存，因此二进制串左移移位即是自相加，这样的末尾会是0。关键就在于判断首位是1还是0，用一个移位寄存器保存下来，如果是1在末尾加1即可。

3. 第二是注意BR控制指令的用法和与CC标识置位的搭配。如何用这两个作一个判断、循环和跳转，以及在代码的什么位置使用这个指令是一个难点，需要清楚SetCC会在何时发生。

4. 本实验所用的方法是比较简单的方法，也是比较通用的方法，时间复杂度为O(N)，仅与移位翻转的位数N有关，代码也非常简短，因此综合来看这种方法还是比较好的。当然，对于16bit pattern来说，移位翻转16次是它的周期，因此还可以作进一步的优化。还有一些其他的算法，比如直接赋值，将第n位值赋给n-1位，这种方法也是可行的，时间复杂度虽然相同，但是单次操作调用的寄存器会更多一些，逻辑上也更加复杂，不是通用的做法，在此就没有尝试。

5. 本次实验实现了任意位的左移翻转，结果是正确的，实验成功。实现了任意位的左移翻转后，就可以实现任意位的右移翻转，右移翻转1位，就相当于左移翻转15位。