

中国科学技术大学计算机学院  
《计算机系统概论》实验报告



实验题目: BINARY DIVISOR

学生姓名: 王章瀚

学生学号: PB18111697

完成日期: 2019 年 11 月 19 日

计算机实验教学中心制

2019 年 09 月

## 1 实验要求

对给定有符号操作数，完成一个 1-bit 的右移算法。

## 2 设计思路

### 2.1 思路一：移位寄存器法

对于右移的操作，可以使用移位寄存器的思路，将最高位补到最末位（第一次直接替换最后一位，因为右移后它不会存在的），并作 15 次左移，最后补足最高位来完成。

以 4bit 有符号数右移为例，如：1101  $\rightarrow$  1011  $\rightarrow$  0110  $\rightarrow$  最后补最高位成为 1110

用下表辅助说明：

	执行前：R0	说明	执行后：R0
第 1 步	1101	$R0[4]==1$	1101
第 2 步	1101	$R0[4]==1$	1011
第 3 步	1011	$R0[4]==0$	0110
第 4 步	0110	R0 原为负数	1110

### 2.2 思路二：直接赋值法

也可以尝试直接把第 n 位的值赋给第 (n-1) 位。

整体思路是利用两个寄存器，比如 R1 最开始保存 x0001，R2 最开始保存 x0002。然后将 R0 与 R2 取与，可以知道 R0 上对应位是否为 1，如果是 1，则  $R4 += R1$ 。然后将 R1, R2 左移，并循环该操作。直到 R2 为 0，则完成，最后确认最高位即可。

例如：

	R0	R1	R2	R4	说明
第 1 步	1101	0001	0010	0000	$R0[1]==0$
第 2 步	1101	0010	0100	0010	$R0[2]==1$
第 3 步	1101	0100	1000	0110	$R0[3]==1$
第 4 步	1101	1000	0000	1110	R0 为负数

后面 3.2.2 中还会提到，这种方法中，由于 R1,R2 的使用有一定重复

性，可以将两次循环并成 1 次（即 R1 和 R2 的作用不断互换），充分利用 R1,R2 的值的相关性来改进算法。

此外，如果要追求极端的执行语句少，由于只有 16 位，可以复制 15 遍代码。经过测试，这样执行完 x0000 的右移，只需要 58 条指令。但这显然是极为不妥的，因此它占用了过大的内存空间。

### 3 关键代码讲解

#### 3.1 思路一：移位寄存器法

过程即按照前述思路进行。代码已注释好。

最开始将 R0 的符号记录在 R3 里，而最后设置符号时，考虑了 R0[15] 现在为 0，且 R3 记录了原本为 1，才需要令 R0[15] 为 1。

这个代码执行对 x8a9c 的右移需要 75 条指令。

对应代码如下：

```

1      0011000000000000      ;      .ORIG    x3000
2      0010001000010001      ;      LD  R1, Nx000E  ; serve as a counter
3      0010011000010001      ;      LD  R3, NxFFFE  ; set R[0] to 0
4      0101000000000011      ;      AND R0, R0, R3
5      0010011000010000      ;      LD  R3, Nx8000
6      0101011000000011      ;      AND R3, R0, R3
7      0000011000000001      ;      BRzp LOOP      ; check the sign of R0
8      0001000000100001      ;      ADD R0, R0, #1
9      0001000000000000      ; LOOP  ADD R0, R0, R0  ; shift left
10     0000011000000001      ;      BRzp ZP
11     0001000000100001      ;      ADD R0, R0, #1  ; if R[15] is 1, R[0] <- 1
12     0001001001111111      ; ZP   ADD R1, R1, #-1
13     0000001111111011      ;      BRp  LOOP
14     0101000000000000      ;      AND R0, R0, R0
15     0000100000000011      ;      BRn  OK
16     0101011011000011      ;      BRzp R3, R3, R3
17     0000011000000001      ;      BRzp OK
18     0001000000000011      ;      ADD R0, R0, R3  ; these set the sign of R0
19     1111000000100101      ; OK   HALT
20     0000000000001110      ;      .FILL   x000E
21     1111111111111110      ;      .FILL   xFFFE
22     1000000000000000      ;      .FILL   x8000
23
24

```

移位寄存器法 (.hex)

## 3.2 思路二：直接赋值法

### 3.2.1 原始思路

过程相对简单，即按照前述思路进行。代码已注释好。

其中比较不易理解的地方是最后的设置符号。由于到了最后 R1 必然是 x8000，因此可以将它与 R0 取与，这样 R1 就得到了 R0[15] 的值，然后往 R4 加即可

这个代码执行对 x8a9c 的右移需要 88 条指令。

```
1 0011000000000000 ; begin at x3000.
2 0010001000001011 ; LD R1, Nx0001
3 0010010000001011 ; LD R2, Nx0002
4 0101100100100000 ; AND R4, R4, #0
5 0101011000000010 ;LOOP AND R3, R0, R2 ; labeled LOOP
6 0000010000000001 ; BRz ZERO
7 0001100100000001 ; ADD R4, R4, R1
8 0001001001000001 ;ZERO ADD R1, R1, R1 ; labeled ZERO
9 0001010010000010 ; ADD R2, R2, R2
10 0000101111111010 ; BRnp LOOP
11 0101001000000001 ; AND R1, R0, R1
12 0001000100000001 ; ADD R0, R4, R1 ; to set the sign
13 1111000000100101 ; HALT
14 0000000000000001 ;Nx0001 .FILL x0001 ; labeled Nx0001
15 0000000000000010 ;Nx0002 .FILL x0002 ; labeled Nx0002
16
```

直接赋值法 (.bin)

### 3.2.2 改进算法

但是上述思路执行的语句还比较多，观察可以发现，R1 和 R2 的功能有些重复，可以想办法复用来减少语句数。我的解决方案是：将两次循环混成一次，以轮流使用 R1 和 R2 的功能。

这样将需要的指令数减少到了 73。

因为只是自己尝试着做的拓展实验，就没按照 hex 来写，而是为方便实现，用了汇编：

```
1 .ORIG x3000
2 LD R1, Nx0001
3 LD R2, Nx0002
4 AND R4, R4, #0 ;一些初始化
5 ; 这一段LOOP将前述方法两次循环综合为一次
```

```

6      ; 这样做的好处在于，每个运算阶段中R1或R2
7      ; 的自加，有且仅有一个需要做，省去了一条
8      ; 指令。
9      LOOP    AND      R3, R0, R2    ; 循环体开始
10     BRz     ZERO1
11     ADD     R4, R4, R1    ; 若该位是1，则对应位加1
12     ZERO1   ADD      R1, R2, R2
13     BRz     FINAL        ; 判断是否结束
14     AND     R3, R0, R1
15     BRz     ZERO2
16     ADD     R4, R4, R2    ; 若该位是1，则对应位加1
17     ZERO2   ADD      R2, R1, R1
18     BRnp    LOOP        ; 循环体结束
19
20     FINAL   AND      R2, R0, R2
21     ADD     R0, R4, R2
22
23     HALT
24
25     Nx0001  .FILL     x0001
26     Nx0002  .FILL     x0002
27     .END
28

```

直接赋值法改良 (.asm)

## 4 调试分析

### 4.1 测试数据选取

这两种方法看起来容易出问题的就是正负号的判断。

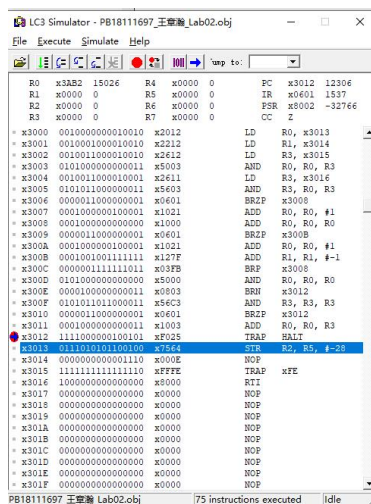
至于其他的运算上，都是经过一定的判断决定是否要不要加 1，要不要移位等，并且除了首尾操作略有不同外，中间的操作都是一样的，如果有问题，必然已经呈现。所以一般一个输入成功，基本上其他的也不会失败。

综合上述原因考虑，可以用 x8a9c 和 x7564 一正一负及 x0000 三个数据来测试。



## 4.3 测试数据: x7564

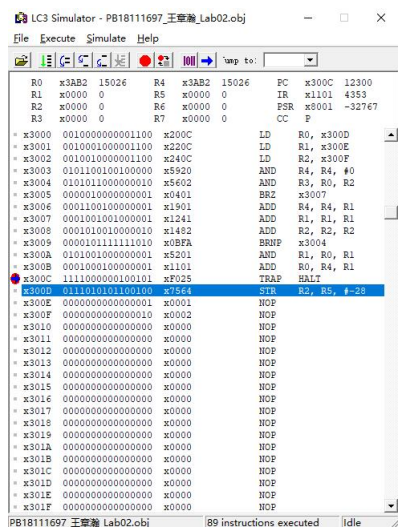
### 4.3.1 移位寄存器法



```
LC3 Simulator - PB18111697_王章瀚_Lab02.obj
File Execute Simulate Help
R0 x3AB2 15026 R4 x0000 0 PC x3012 12304
R1 x0000 0 R5 x0000 0 IR x0401 1537
R2 x0000 0 R6 x0000 0 FSR x3002 -32766
R3 x0000 0 R7 x0000 0 CC Z
+ x3000 0010000000000010 x2012 LD R0, x3013
+ x3001 00100010000010010 x2212 LD R1, x3014
+ x3002 00100110000010010 x2412 LD R2, x3015
+ x3003 0101000000000011 x5003 AND R0, R0, R3
+ x3004 00100110000010001 x2611 LD R3, x3016
+ x3005 0101011000000011 x5603 AND R2, R0, R3
+ x3006 0000011000000001 x0601 BRZF x3006
+ x3007 0001000000100001 x1021 ADD R0, R0, #1
+ x3008 0001000000000000 x1000 ADD R0, R0, R0
+ x3009 0000011000000001 x0601 BRZF x3008
+ x300A 0001000000100001 x1021 ADD R0, R0, #1
+ x300B 0001001001111111 x127F ADD R1, R1, #-1
+ x300C 0000001111111111 x33FB BRZ x300C
+ x300D 0101000000000000 x5000 AND R0, R0, R0
+ x300E 0000100000000011 x0803 BRH x3012
+ x300F 0101011011000011 x56C3 AND R3, R3, R3
+ x3010 0000011000000001 x0601 BRZF x3012
+ x3011 0001000000000011 x1003 ADD R0, R0, R3
+ x3012 1111000000100101 xF025 TRAP HALT
+ x3013 0111010101010101 x1274 STJ R2, R5, #-28
+ x3014 0000000000001110 x000E NOP
+ x3015 1111111111111110 xFFFF TRAP xFE
+ x3016 1000000000000000 x0000 RTI
+ x3017 0000000000000000 x0000 NOP
+ x3018 0000000000000000 x0000 NOP
+ x3019 0000000000000000 x0000 NOP
+ x301A 0000000000000000 x0000 NOP
+ x301B 0000000000000000 x0000 NOP
+ x301C 0000000000000000 x0000 NOP
+ x301D 0000000000000000 x0000 NOP
+ x301E 0000000000000000 x0000 NOP
+ x301F 0000000000000000 x0000 NOP
PB18111697_王章瀚_Lab02.obj 75 instructions executed idle
```

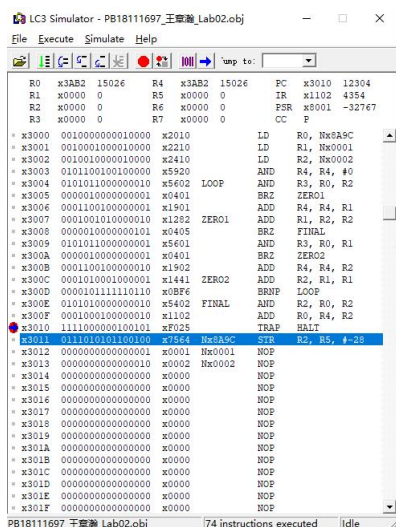
图 4: 移位寄存器法: x7564

### 4.3.2 直接赋值法



```
LC3 Simulator - PB18111697_王章瀚_Lab02.obj
File Execute Simulate Help
R0 x3AB2 15026 R4 x3AB2 15026 PC x300C 12300
R1 x0000 0 R5 x0000 0 IR x1101 4353
R2 x0000 0 R6 x0000 0 FSR x3001 -32767
R3 x0000 0 R7 x0000 0 CC P
+ x3000 0010000000001100 x200C LD R0, x300D
+ x3001 0010001000001100 x220C LD R1, x300E
+ x3002 0010010000001100 x240C LD R2, x300F
+ x3003 0101100100100000 x5920 AND R4, R4, #0
+ x3004 0101011000000010 x5602 AND R3, R0, R2
+ x3005 0000010000000001 x0401 BRZ x3007
+ x3006 0001100100000001 x1501 ADD R4, R4, R1
+ x3007 0001001001000001 x1241 ADD R1, R1, R1
+ x3008 0001001001000001 x1482 ADD R2, R2, R2
+ x3009 0000101111111010 x08FA BRNP x3004
+ x300A 0101001000000001 x3201 AND R1, R0, R1
+ x300B 0001000100000001 x1101 ADD R0, R4, R1
+ x300C 1111000000100101 xF025 TRAP HALT
+ x300D 0111010101010101 x1274 STJ R2, R5, #-28
+ x300E 0000000000000001 x0001 NOP
+ x300F 0000000000000010 x0002 NOP
+ x3010 0000000000000000 x0000 NOP
+ x3011 0000000000000000 x0000 NOP
+ x3012 0000000000000000 x0000 NOP
+ x3013 0000000000000000 x0000 NOP
+ x3014 0000000000000000 x0000 NOP
+ x3015 0000000000000000 x0000 NOP
+ x3016 0000000000000000 x0000 NOP
+ x3017 0000000000000000 x0000 NOP
+ x3018 0000000000000000 x0000 NOP
+ x3019 0000000000000000 x0000 NOP
+ x301A 0000000000000000 x0000 NOP
+ x301B 0000000000000000 x0000 NOP
+ x301C 0000000000000000 x0000 NOP
+ x301D 0000000000000000 x0000 NOP
+ x301E 0000000000000000 x0000 NOP
+ x301F 0000000000000000 x0000 NOP
PB18111697_王章瀚_Lab02.obj 89 instructions executed idle
```

图 5: 直接赋值法: x7564



```
LC3 Simulator - PB18111697_王章瀚_Lab02.obj
File Execute Simulate Help
R0 x3AB2 15026 R4 x3AB2 15026 PC x3010 12304
R1 x0000 0 R5 x0000 0 IR x1102 4354
R2 x0000 0 R6 x0000 0 FSR x3001 -32767
R3 x0000 0 R7 x0000 0 CC P
+ x3000 0010000000001000 x2010 LD R0, x30A9C
+ x3001 0010001000001000 x2210 LD R1, x30001
+ x3002 0010010000001000 x2410 LD R2, x30002
+ x3003 0101100100100000 x5920 AND R4, R4, #0
+ x3004 0101011000000010 x5602 LOOP AND R3, R0, R2
+ x3005 0000010000000001 x0401 BRZ ZERO1
+ x3006 0001100100000001 x1501 ADD R4, R4, R1
+ x3007 0001001001000001 x1282 ZERO1 ADD R1, R2, R2
+ x3008 0000010000000101 x0405 BRZ FINAL
+ x3009 0101011000000001 x5601 AND R3, R0, R1
+ x300A 0000010000000001 x0401 BRZ ZERO2
+ x300B 0001100100000010 x1502 ADD R4, R4, R1
+ x300C 0001000010000001 x1441 ZERO2 ADD R2, R1, R1
+ x300D 0000101111110110 x08FE BRNP LOOP
+ x300E 0101010000000010 x5402 FINAL AND R2, R0, R2
+ x300F 0001000010000010 x1102 ADD R0, R4, R2
+ x3010 1111000000100101 xF025 TRAP HALT
+ x3011 0111010101010101 x1274 STJ R2, R5, #-28
+ x3012 0000000000000001 x0001 Hx0001 NOP
+ x3013 0000000000000010 x0002 Hx0002 NOP
+ x3014 0000000000000000 x0000 NOP
+ x3015 0000000000000000 x0000 NOP
+ x3016 0000000000000000 x0000 NOP
+ x3017 0000000000000000 x0000 NOP
+ x3018 0000000000000000 x0000 NOP
+ x3019 0000000000000000 x0000 NOP
+ x301A 0000000000000000 x0000 NOP
+ x301B 0000000000000000 x0000 NOP
+ x301C 0000000000000000 x0000 NOP
+ x301D 0000000000000000 x0000 NOP
+ x301E 0000000000000000 x0000 NOP
+ x301F 0000000000000000 x0000 NOP
PB18111697_王章瀚_Lab02.obj 74 instructions executed idle
```

图 6: 改良的直接赋值法: x7564

## 4.4 测试数据: x0000

### 4.4.1 移位寄存器法

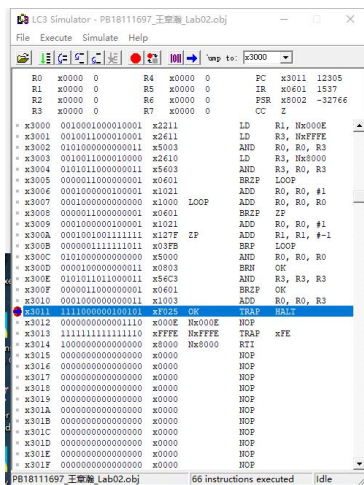


图 7: 移位寄存器法: x0000

### 4.4.2 直接赋值法

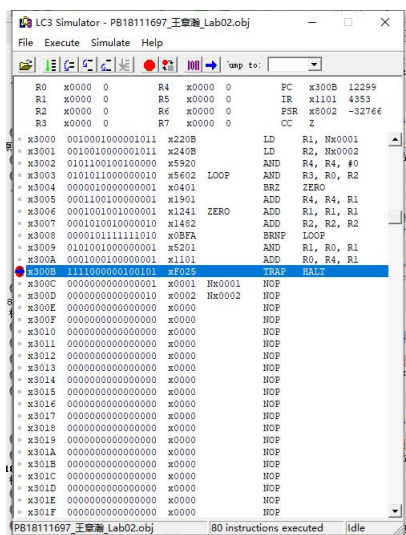


图 8: 直接赋值法: x0000

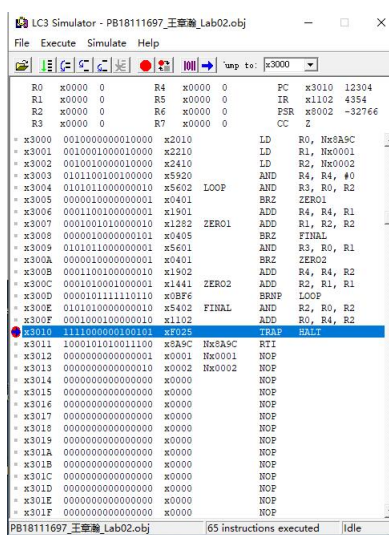


图 9: 改良的直接赋值法: x0000



## 5 实验总结

本次实验对有符号 16 位操作数实现了右移操作，虽然看起来比较简单，但要实现出一个高效的算法实在不容易。

就本人写的这几个代码来看，综合代码长度和执行效率，应该是**改良的直接赋值法**最佳。

除了上述绞尽脑汁想出来的算法，我还听说过有人使用了一些表来实现等。总之实现右移的方法多种多样，但不论如何，似乎都不能避免这 15 到 16 次的循环，而每个循环也不容易压缩到小于 5 条指令。这可能也就是计算机不易做除法的原因之一吧。