REPORT LAB01

PB19010450 和泳毅

一、实验要求

用LC-3机器语言编写一个程序，将一个16位的值向**左旋转**2位。

将一个位模式向**左旋转**n位类似于**左移位**n位，但移位前的前n位最终会作为位移后的后n位。

具体内容：

- 初始位模式在内存位置**x3100**。
- 旋转量存储在内存位置**x3101**。
- 左旋转结果存储在内存位置**x3102**。
- 程序应该在内存位置 **x3000** 开始。

例子：

1. 内存位置**x3100**包含值`1101 0001 0000 1000`；
2. 内存位置**x3101**包含值`0000 0000 0000 0010`(十进制2)；
3. 那么程序需要左旋转2位，并将结果`0100 0100 0010 0011`在内存位置**x3102**。

二、实验设计

左旋转为左位移的进阶操作，唯一不同点在于：

- 左位移可以实现位运算左移操作，但前n位会溢出，平移后的后n位全为0。
- 左旋转前n位会作为移动完的后n位。

所以先考虑左平移操作，再解决溢出问题。

1. 左位移操作

   将一个16位的值与自身相加，其结果较原值左移1位。

   例如：

   ```
     1101 0001 0000 1000
   + 1101 0001 0000 1000
   -----------------------
     1010 0010 0001 0000
   ```

2. 溢出问题的解决

   为方便描述，记原值为A，左平移后的值为B，左旋转的结果为C。

   左平移对原值的副本操作，保留原值。如果左旋转n位得到B，B相对于A溢出了前n位，并且后n位为0。那么只需要判断A前n位的值，我们称为补偿值，加至B，即可解决溢出问题得到C。

   判断A前n位的值，运用AND操作判断某位是否为1，当A的第n位为1的时候，B+`0000 0000 0000 0001`。此后再对第n-1位判断、相加操作，循环直到进行到A到第一位结束，得到C。

   例如：

```
A = 1101 0001 0000 1000 旋转2位
B = 0100 0100 0010 0000

  0100 0100 0010 0000       0100 0100 0010 0001
+ 0000 0000 0000 0001     + 0000 0000 0000 0010
----------------------  ----------------------
  0100 0100 0010 0001       0100 0100 0010 0011

C = 1010 0010 0001 0011
```

三、代码编写

1. 运用到的指令

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BR	0000				n	z	p	PCoffset9								
ADD	0001				DR			SR1			1	imm5				
ADD	0001				DR			SR1			000			SR2		
ST	0011				SR			PCoffset9								
AND	0101				DR			SR1			1	imm5				
AND	0101				DR			SR1			000			SR2		
LDR	0110				DR			BaseR			offset6					
STR	0111				DR			BaseR			offset6					
NOT	1001				DR			SR			111111					
LEA	1110				DR			PCoffset9								
TRAP	1111				0000						trapvect8					

2. 起始内存**x3000**

```
0011 0000 0000 0000 ; 从x3000开始
```

3. 寄存器初始化

   AND指令使目标内容与`0000 0000 0000 0000`做与运算再存于自身即可清零。

```
0101 010 010 1 00000 ; 初始化R2
0101 011 011 1 00000 ; 初始化R3
0101 100 100 1 00000 ; 初始化R4
0101 101 101 1 00000 ; 初始化R5
```

4. 对地址**x3100**的内容（初始值）的读取

   先把地址**x3100**作为值存入寄存器，再使用LDR取偏移量为0读取**x3100**的内容。

```
1110 010 011111011 ; 把数x3100存到R2
0110 011 010 000000 ; 把x3100的内容放入R3
```

5. 对地址**x3101**的内容（旋转量）的读取

   使用ADD使得值**x3100**增一，再使用LDR取偏移量为0读取**x3101**的内容。

```
0001 010 010 1 00001 ; R2++ = x3101
0110 100 010 000000 ; 把x3101的内容放入R4
```

6. 对原值进行左平移

   使用ADD与`0000 0000 0000 0000`相加实现复制。使用ADD与`1111 1111 1111 1111`相加实现计算器减一。

```
0001 101 011 1 00000 ; 复制R3->R5，作为原值副本
0001 000 100 1 00000 ; R4 + 1->R0，计数器
0001 101 101 000 101 ; R5 + R5->R5
0001 000 000 1 11111 ; R0--
0000 101 111111101 ; 计数器未到0，循环至x300A
```

7. 溢出值的处理

   为了判断某位为1/0，需要置出只有第n位为1的值，利用`0000 0000 0001`左平移思想循环实现。

   若为1则使得平移后的值与补偿值相加，此后AND判断值与补偿值同时自加左移，持续操作直到判断至原值的首位，此时AND判断值应该为`1000 0000 0000 0000`，补偿值只有倒数第n位为1。

   R0计数器先用于置出AND判断值，后用于置出补偿值。R7计数器用于整个溢出值处理的过程循环的计数。

```
0101 000 000 1 00000 ; 初始化R0
0001 000 000 1 01111 ; R0 = 15，计数器
0101 110 110 1 00000 ; 初始化R6
0001 110 110 1 00001 ; R6 = 1
1001 111 100 111111 ;
0001 111 111 1 00001 ; R7 + 1，上行取反，此行加一，相当于取负数，计数器
0001 110 110 000 110 ; R6 + R6->R6
0001 000 000 1 11111 ; R0--
0001 001 000 000 111 ; R0 + R7->R1
0000 101 111111100 ; 计数器未到0，循环至x3013
0101 000 000 1 00000 ; 初始化R0
0001 000 000 1 00001 ; R0 = 1
0001 110 110 000 110 ; R6 + R6->R6
0101 001 011 000 110 ; AND判断某位是否为1
0000 010 000000001 ; 如果是0不加补偿值
0001 101 101 000 000 ; 如果是1，补偿值加至目标数
0001 000 000 000 000 ; R0 + R0->R0，补偿值
0001 111 111 1 00001 ; R7++
0000 100 111111001 ; 计数器未到0，循环至x3019
```

8. 结果存入**x3102**

   使用STR将结果存入**x3101 + 1 = x3102**

```
0111 101 010 000001 ; 存入R5到地址x3102
1111 0000 00100101 ; 程序终止
```

9. 回顾寄存器使用情况

R0	计数器，补偿值
R1	BR临时判断值
R2	X3100值
R3	初始值
R4	旋转量
R5	左平移后的值，最终结果
R6	AND判断值
R7	计数器

四、结果测试

1. 实验要求的例子测试

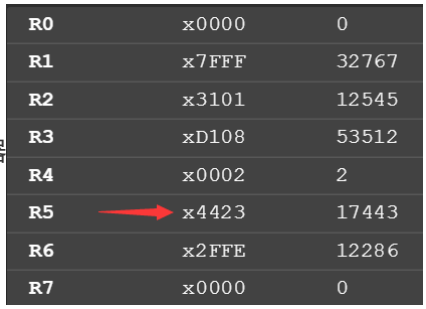
   `1101 0001 0000 1000`(xD108) 左旋转2位 = `0100 0100 0010 0011`(x4423)

   在**x3100**读入`1101 0001 0000 1000`(xD108)，**x3101**读入`0000 0000 0000 0010`(x0002)

   结果

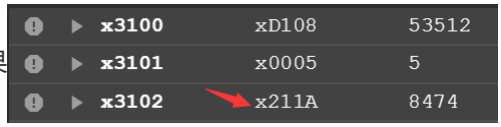
   寄存器

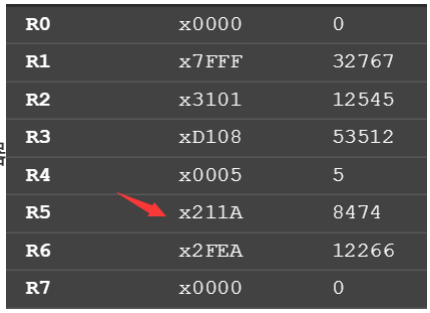
2. 改变旋转量

   `1101 0001 0000 1000`(xD108) 左旋转5位 = `0010 0001 0001 1010`(x211A)

   结果

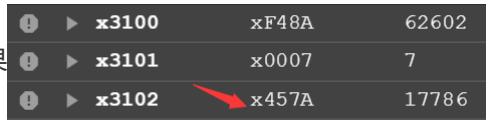
   寄存器

3. 改变初始值

   `1111 0100 1000 1010`(xF48A) 左旋转7位 = `0100 0101 0111 1010`(x457A)

   结果

寄存器

R0	x0000	0
R1	x7FFF	32767
R2	x3101	12545
R3	xF48A	62602
R4	x0007	7
R5	x457A	17786
R6	x2FFC	12284
R7	x0000	0

4. 旋转量界限测试

   经过对旋转量的测试发现，当旋转量属于[1,14]时都能正确得到结果，而旋转量属于(−∞,0]∪[15,+∞)时，由于代码计数器的设计缺陷无法得到正确结果。可以做以下改进：

   （1）读取旋转量的时候先判断是否为0和16，如果是则在**x3102**直接存入初始值，结束程序。

   （2）再判断是否大于16，如果大于16则与16作差（16为一旋转周期），直到新的旋转量属于[0,16]。

   （3）再判断是否小于0，如果小于0即可认为是右旋转，与16作和，直到新的旋转量属于[0,16]（例如-1表示右旋转1位<=>左旋转15位）。

5. 初始值的特殊情况

   对于`0000 0000 0000 0000`和`1111 1111 1111 1111`，无论如何旋转都不变，可以提前结束程序。而此代码虽然对他们运算结果正确，但复杂度偏高。可以开始用AND对二者进行判断，如果是则在**x3102**直接存入初始值，结束程序。

五、实验思考与改进

1. 实验过程中发现程序结束的时候寄存器R0 R1 R6的内容会发生奇怪地改变。经过断点测试，发现在最后TRAP前断点，R0 R1 R6 的内容为正确应有内容，而在执行完TRAP_HALT后发生变化，推测TRAP指令在R0 R1 R6上有操作。
2. 实验过程中发现程序结束的时候PC停止在x0263而不是TRAP指令所在地址x3021。找到x0263地址发现对于存储内容为TRAP_HALT。

x0263	x0FF9	4089	*BRnzp TRAP_HALT*

3. 写完代码后，发现极其冗杂，认为左平移和溢出处理或许可以同时进行。便对此进行一定地改进：
   - 保存初始值副本，对副本进行左平移，同时判断初始值首位为1/0，如果是1，则左平移后的值ADD `0000 0000 0000 0001`；

- 保存上一步结果副本，重复进行上述操作知道满足旋转量。

  （1）**核心代码**

```
0001 101 011 1 00000 ; 复制R3->R5
0101 000 000 1 00000 ; 初始化R0
0001 000 100 1 00000 ; R4->R0
0101 001 101 000 110 ; R6存储AND判断值，判断某位是否为1
0001 101 101 000 101 ; R5 + R5->R5左移
0001 001 001 1 00000 ; R1
0000 010 000000001 ; 如果是0不加补偿值
0001 101 101 1 00001 ; R5+1->R5
0001 000 000 1 11111 ; R0--
0000 101 111111001 ; 计数器未到0，循环
```

  （2）**结果测试**

  - 实验要求的例子测试

`1101 0001 0000 1000`(xD108) 左旋转2位 = `0100 0100 0010 0011`(x4423)

结果

x3100	初始值	xD108	53512
x3101	旋转量	x0002	2
x3102	结果	x4423	17443

寄存器

R0	x0000	0
R1	x7FFF	32767
R2	x3101	12545
R3	xD108	53512
R4	x000E	14
R5	x3442	13378
R6	x8000	32768
R7	x0000	0

- 改变旋转量

`1101 0001 0000 1000`(xD108) 左旋转5位 = `0010 0001 0001 1010`(x211A)

结果

x3100		xD108	53512
x3101		x0005	5
x3102		x211A	8474

寄存器

R0	x0000	0
R1	x7FFF	32767
R2	x3101	12545
R3	xD108	53512
R4	x0005	5
R5	x211A	8474
R6	x2FF6	12278
R7	x0000	0

- 改变初始值

`1111 0100 1000 1010`(xF48A) 左旋转7位 = `0100 0101 0111 1010`(x457A)

结果

x3100		xF48A	62602
x3101		x0007	7
x3102		x457A	17786

寄存器

R0	x0000	0
R1	x7FFF	32767
R2	x3101	12545
R3	xF48A	62602
R4	x0007	7
R5	x457A	17786
R6	x2FF2	12274
R7	x0000	0

- 旋转量界限测试

对计数器重新设计，使得当旋转量属于[0,16]时都可以得到正确答案（弥补原版不能旋转0、15、16位的缺陷）。

x3100		xF48A	62602
x3101		x0010	16
x3102		xF48A	62602

x3100		xF48A	62602
x3101		x0000	0
x3102		xF48A	62602

并且对于属于[17,+∞)的旋转量，本改进代码依然可以正确运行弥补原版不能旋转大于16位的缺陷）。

x3100		xD108	53512
x3101		x0012	18
x3102		x4423	17443

x3100		xD108	53512
x3101		x0052	82
x3102		x4423	17443

4. 最后，进行进一步改进，主要考虑复杂度问题。
   - 注意到我们只需要对首位1/0进行判断，因为首位符号具有特殊性，代表了正数与负数，可以直接通过判断正负来判断首位1/0;
   - 初版代码编写为了避免错误，将运用到的寄存器全部初始化清零。后来发现对于从内存中读入的，用于保存结果的寄存器，我们不需要进行清零；
   - 善用基址偏移寻址LDR进行对连续地址的读取。

（1）**核心代码**

```
0001 101 011 1 00000 ; 复制R3->R5
0001 011 011 000 101 ; R3 + R3->R3，左移
0001 101 101 1 00000 ; 读取R5
0000 001 000000001 ; 如果首位是0不加补偿值
0001 011 011 1 00001 ; R3+1->R3
0001 100 100 1 11111 ; R4--
0000 101 111111001 ; 计数器未到0，循环
```

（2）**结果测试**

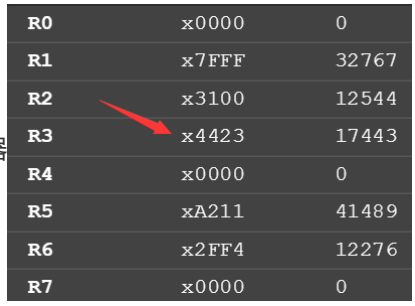
- 实验要求的例子测试

  `1101 0001 0000 1000`(xD108) 左旋转2位 = `0100 0100 0010 0011`(x4423)

  结果

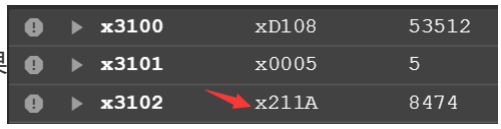
  寄存器

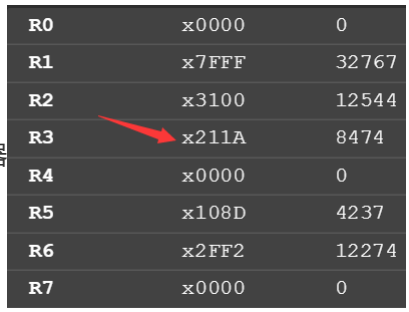
- 改变旋转量

  `1101 0001 0000 1000`(xD108) 左旋转5位 = `0010 0001 0001 1010`(x211A)

  结果

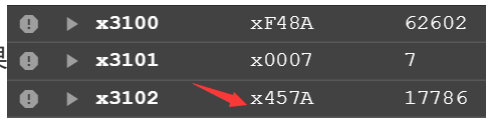
  寄存器

- 改变初始值

  `1111 0100 1000 1010`(xF48A) 左旋转7位 = `0100 0101 0111 1010`(x457A)

  结果

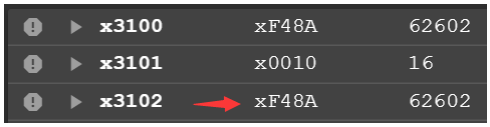
  

寄存器

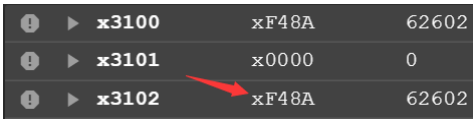
R0	x0000	0
R1	x7FFF	32767
R2	x3100	12544
R3	x457A	17786
R4	x0000	0
R5	x22BD	8893
R6	x2FF0	12272
R7	x0000	0

- 旋转量界限测试

  当旋转量属于[0,16]时都可以得到正确答案。



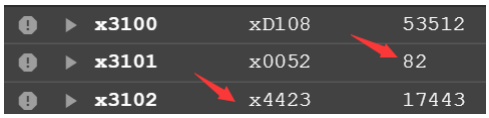
x3100	xF48A	62602
x3101	x0010	16
x3102	xF48A	62602



x3100	xF48A	62602
x3101	x0000	0
x3102	xF48A	62602

  当旋转量属于[17,+∞)时都可以得到正确答案。

x3100	xD108	53512
x3101	x0012	18
x3102	x4423	17443



x3100	xD108	53512
x3101	x0052	82
x3102	x4423	17443

六、实验总结

第一次用LC-3机器语言编写程序，感觉十分复杂晦涩。但通过简单的基本操作却可以得到不显然的结果，使得简单的指令具有强大的隐性功能。同时发现，对于同一个目的，可以使用不同的指令、不同的寄存器来实现，所以每个人的代码可能大有不同，使得问题的解决方案具有多样性。这同时也意味着一个问题有着相对最优方案，它可能有着最小的复杂度、最简短的表达、最清晰的思路或者最短的行数。本人也尝试着将初版代码进行了简单的优化，将左平移和溢出处理在同一个循环下实现，使得代码行数大大减少，复杂度也得到降低。这是一件十分有趣的事情，但由于时间有限，本人对初次编写LC-3机器语言程序的结果与收获已经满足，不再进一步优化（第三版代码可以对通用性进行改进）。

七、RISC-V汇编重写

思路不变，左位移+补偿值=左旋转。由于时间有限，使用在网上寻找到的在线编译器*venus*。

> 在线编译器网站：http://venus.cs61c.org/
>
> Github：https://github.com/kvakil/venus
>
> **venus**
>
> venus is a RISC-V instruction set simulator built for education.
>
> **Using venus**
>
> venus is available online.
>
> **Features**
>
> - RV32IM
> - Single-step debugging with undo feature
> - Breakpoint debugging
> - View machine code and original instructions side-by-side
> - Several `ecall`s: including `print` and `sbrk`
> - Memory visualization

1. 使用到的指令

- **andi rd,rs1,immediate**

  与立即数(*And Immediate*)。 x[rd] = x[rs1] & sext(immediate)

  把符号位扩展的立即数和寄存器x[*rs1*]上的值进行位与，结果写入x[*rd*]

- **add rd,rs1,rs2**

  加运算(*Add*) x[rd] = x[rs1] + x[rs2]

  把寄存器 x[*rs2*]加到寄存器 x[rs1]上，结果写入x[*rd*]。忽略算术溢出。

- **bge rs1,rs2,offset**

  大于等于时分支(*Branch if Greater Than or Equal*) if (rs1 >= re2) pc += sext(offset)

  若寄存器x[*rs1*]的值大于等于寄存器x[*rs2*]的值（均视为二进制补码），把*pc*的值设为当前值加上符号位扩展的偏移offset。

2. 核心代码

```
andi x1,x1,0    #x1清零作为判断值
addi x4,x3,0    #复制
add x3,x3,x3    #x3+x3->x3
bge x4,x1,8     #x4>0则跳过下一步
addi x3,x3,1    #x3++
addi x2,x2,-1   #x2--旋转量计数器
bge x2,x1,-20   #x2>0循环
```

3. 寄存器的使用

ra(x1)	AND判断值
sp(x2)	旋转量
gp(x3)	初始值
tp(x4)	中间值，最终结果

4. 结果测试

初始：

结果：

八、附录

完整代码：

初版：

```
0011 0000 0000 0000 ; 程序从x3000开始
0101 010 010 1 00000 ; 这里真正地从x3000开始,初始化R2
```

```
0101 011 011 1 00000 ; 初始化R3
0101 100 100 1 00000 ; 初始化R4
0101 101 101 1 00000 ; 初始化R5
1110 010 011111011 ; 把数x3100存到R2
0110 011 010 000000 ; 把地址x3100的内容放入R3
0001 010 010 1 00001 ; R2++ = x3101
0110 100 010 000000 ; 把地址x3101的内容放入R4
0001 101 011 1 00000 ; 复制R3->R5，作为原值副本
0001 000 100 1 00000 ; R4 + 1->R0，计数器
0001 101 101 000 101 ; R5 + R5->R5
0001 000 000 1 11111 ; R0--
0000 101 111111101 ; 计数器未到0，循环至x300A
0101 000 000 1 00000 ; 初始化R0
0001 000 000 1 01111 ; R0 = 15，计数器
0101 110 110 1 00000 ; 初始化R6
0001 110 110 1 00001 ; R6 = 1
1001 111 100 111111 ;
0001 111 111 1 00001 ; R7 + 1，上行取反，此行加一，相当于取负数，计数器
0001 110 110 000 110 ; R6 + R6->R6
0001 000 000 1 11111 ; R0--
0001 001 000 000 111 ; R0 + R7->R1
0000 101 111111100 ; 计数器未到0，循环至x3013
0101 000 000 1 00000 ; 初始化R0
0001 000 000 1 00001 ; R0 = 1
0001 110 110 000 110 ; R6 + R6->R6
0101 001 011 000 110 ; AND判断某位是否为1
0000 010 000000001 ; 如果是0不加补偿值
0001 101 101 000 000 ; 如果是1，补偿值加至目标数
0001 000 000 000 000 ; R0 + R0->R0，补偿值
0001 111 111 1 00001 ; R7++
0000 100 111111001 ; 计数器未到0，循环至x3019
0111 101 010 000001 ; 存入R5到地址x3102
1111 0000 00100101 ; 程序终止
```

第二版:

```
0011 0000 0000 0000 ; 程序从x3000开始
0101 010 010 1 00000 ; 这里真正地从x3000开始,初始化R2
0101 011 011 1 00000 ; 初始化R3
0101 100 100 1 00000 ; 初始化R4
0101 101 101 1 00000 ; 初始化R5
1110 010 011111011 ; 把数x3100存到R2
0110 011 010 000000 ; 把地址x3100的内容放入R3
0001 010 010 1 00001 ; R2++ = x3101
0110 100 010 000000 ; 把地址x3101的内容放入R4
0101 000 000 1 00000 ; 初始化R0
0001 000 000 1 01111 ; R0 = 15，计数器
0101 110 110 1 00000 ; 初始化R6
0001 110 110 1 00001 ; R6 = 1
0001 110 110 000 110 ; R6 + R6->R6，左移，置出AND判断值
0001 000 000 1 11111 ; R0--
0000 101 111111101 ; 计数器未到0，跳至x300C
0001 101 011 1 00000 ; 复制R3->R5
0101 000 000 1 00000 ; 初始化R0
0001 000 100 1 00000 ; R4->R0，计数器
0101 001 101 000 110 ; AND判断某位是否为1
0001 101 101 000 101 ; R5 + R5->R5，左移
```

```
0001 001 001 1 00000 ; 读取R1
0000 010 000000001 ; 如果是0不加补偿值
0001 101 101 1 00001 ; R5+1->R5
0001 000 000 1 11111 ; R0--
0000 101 111111001 ; 计数器未到0，跳至x300A
0111 101 010 000001 ; 存入R5到地址x3102
1111 0000 00100101 ; 程序终止
```

第三版：

```
0011 0000 0000 0000 ; 程序从x3000开始
1110 010 011111111 ; 把数x3100存到R2
0110 011 010 000000 ; 把地址x3100的内容放入R3
0110 100 010 000001 ; 把地址x3101的内容放入R4
0001 101 011 1 00000 ; 复制R3->R5
0001 011 011 000 101 ; R3 + R3->R3，左移
0001 101 101 1 00000 ; 读取R5
0000 001 000000001 ; 如果首位是0不加补偿值
0001 011 011 1 00001 ; R3+1->R3
0001 100 100 1 11111 ; R4--
0000 101 111111001 ; 计数器未到0，循环
0111 011 010 000010 ; 存入R5到地址x3102
1111 0000 00100101 ; 程序终止
```

RISC-V版：

```
main:
    andi x1,x1,0    #x1清零作为判断值
    addi x4,x3,0    #复制
    add x3,x3,x3    #x3+x3->x3
    bge x4,x1,8     #x4>0则跳过下一步
    addi x3,x3,1    #x3++
    addi x2,x2,-1   #x2--旋转量计数器
    bge x2,x1,-20   #x2>0循环
```