实验 1. 熟悉 MIPS 汇编程序开发环境，学习使用 MARs 工具。知道如何查看内存空间分配

一.用汇编程序实现以下伪代码：要求使用移位指令实现乘除法运算。

```
Int main ()
{
Int K,Y;
Int Z[50];
Y=56;
For(k=0;k<50;K++) Z[k]=Y-16*(k/4+210);
}
```

二、、程序设计及分析

1.C 语言分析：

有两个变量是 int 型，一个数组型；还有一个循环执行过程。

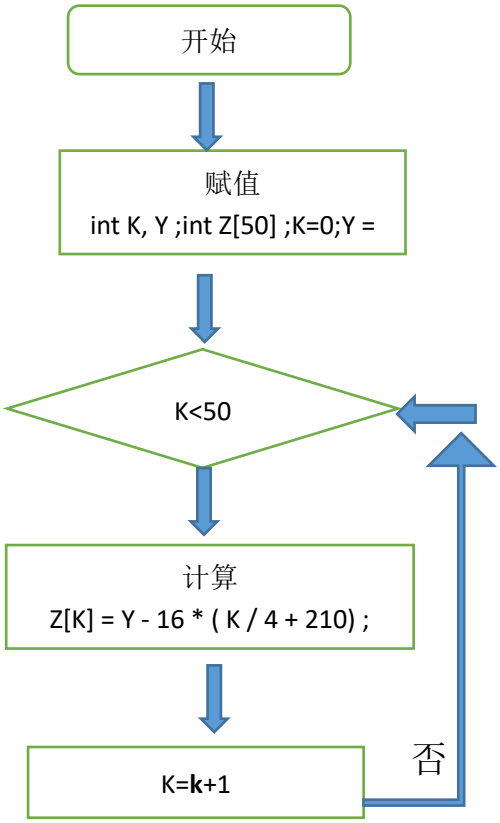
2.汇编程序实现分析：

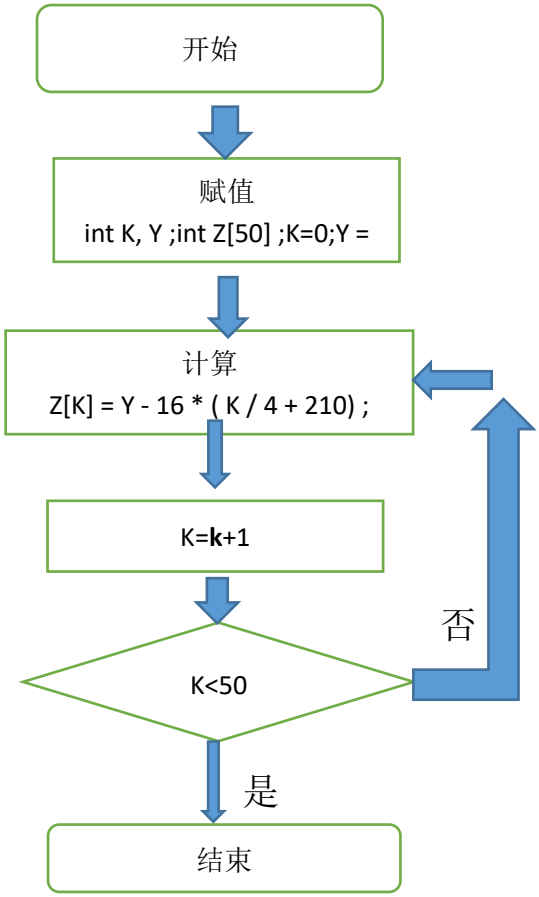
首先需要定义用户数据段，获得一个内存空间作为数组空间。再选定几个寄存器作为 K，Y 以及输出，其中输出输出和 Y 可以合用一个寄存器。

3.设计思路：

分配完空间地址后，最重要的是完成循环控制。循环控制有两个思路：可以是先判断后循环；或者是先循环后判断

即如图





slti $t2,$t0,50 #判断 k 是否于 50

beq $t2,$t3（$t2=1 循环，否则结束。）

slti $t2,$t0,50 #判断 k 是否小于 50，
beq $t2,$0，#是则结束
#否,循环

三、程序实现及调试分析

1. 汇编程序代码实现：

方法一

```
        .data
K:      .word   0
Y:      .word   56
Z:      .space  200             # 给数组Z分配50*4=200个字节的空间
str:    .asciiz " "
        .text
main:
        lw      $s1, K          # $s1代表K
        lw      $s2, Y          # $s2代表Y
        la      $s3, Z          # $s3代表数组Z
loop:
        slti    $t0, $s1, 50    # 若K<50，$t0 = 1，否则$t0 = 0
        beq     $t0, $0, exit   # 若$t0 = 0，即K>=50，则跳转到exit（跳出循环）
        srl     $t1, $s1, 2     # 将$s1右移两位，即$t1 = K / 4
        addi    $t1, $t1, 210   # $t1 = $t1 + 210
        sll     $t1, $t1, 4     # 将$t1左移4位，即$t1 = $t1 * 16
        sub     $t1, $s2, $t1   # $t1 = $s2(Y) - $t1
        sw      $t1, 0($s3)     # 将$t1的值存入$s3(数组Z)的相应位置中
        lw      $a0, 0($s3)     # 输出数组Z当前位置的值
        li      $v0, 1
        syscall
        la      $a0, str        # 输出空格“ ”
        li      $v0, 4
        syscall
        addi    $s3, $s3, 4     # 让数组Z寻址到下一个位置（4个字节）
        addi    $s1, $s1, 1     # K = K + 1
        j       loop            # 跳转到loop，继续循环
exit:
        li      $v0, 10         # 程序终止结束
        syscall
```

方法二

```
        .data
K:      .word   0
Y:      .word   56
Z:      .space  200             # 给数组Z分配50*4=200个字节的空间
str:    .asciiz " "
        .text
main:
        lw      $s1, K          # $s1代表K
        lw      $s2, Y          # $s2代表Y
        la      $s3, Z          # $s3代表数组Z
loop:
        srl     $t1, $s1, 2     # 将$s1右移两位，即$t1 = K / 4
        addi    $t1, $t1, 210   # $t1 = $t1 + 210
        sll     $t1, $t1, 4     # 将$t1左移4位，即$t1 = $t1 * 16
        sub     $t1, $s2, $t1   # $t1 = $s2(Y) - $t1
        sw      $t1, 0($s3)     # 将$t1的值存入$s3(数组Z)的相应位置中
        lw      $a0, 0($s3)     # 输出数组Z当前位置的值
        li      $v0, 1
        syscall
        la      $a0, str        # 输出空格“ ”
        li      $v0, 4
        syscall
        addi    $s3, $s3, 4     # 让数组Z寻址到下一个位置（4个字节）
        addi    $s1, $s1, 1     # K = K + 1
        slti    $t0, $s1, 50    # 若K<50，$t0 = 1，否则$t0 = 0
        beq     $t0, $0, exit   # 若$t0 = 0，即K>=50，则跳转到exit（跳出循环）
        j       loop            # 跳转到loop，继续循环
exit:
        li      $v0, 10         # 程序终止结束
        syscall
```

2.调试过程

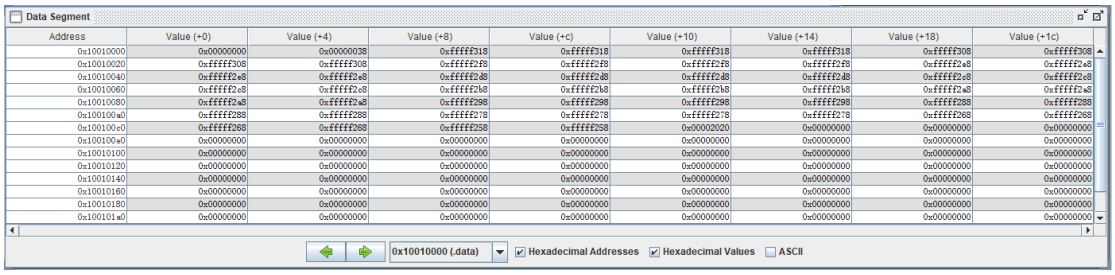
编写程序：详细见代码

装载程序

如果没有错误，便运行。

运行之后点击不同的窗口便可得到我们想要的结果。具体详细结果如下图

内存占用情况映像

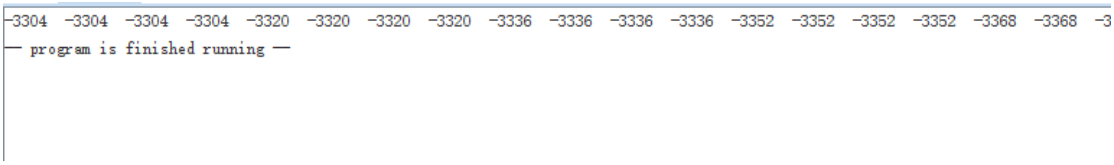


分析：数组地址

数据段内存映像
表格如下（数值都采用 16 进制）

内存地址（16 进制）	变量名	值	内存地址（16 进制）	变量名	值
0x10010008	**Z[0]**	**0xfffff318**	**0x1001006c**	**Z[25]**	**0xfffff2b8**
0x1001000c	**Z[1]**	**0xfffff318**	**0x10010070**	**Z[26]**	**0xfffff2b8**
0x10010010	**Z[2]**	**0xfffff318**	**0x10010074**	**Z[27]**	**0xfffff2b8**
0x10010014	**Z[3]**	**0xfffff318**	**0x10010078**	**Z[28]**	**0xfffff2a8**
0x10010018	**Z[4]**	**0xfffff308**	**0x1001007c**	**Z[29]**	**0xfffff2a8**
0x1001001c	**Z[5]**	**0xfffff308**	**0x10010080**	**Z[30]**	**0xfffff2a8**
0x10010020	**Z[6]**	**0xfffff308**	**0x10010084**	**Z[31]**	**0xfffff2a8**
0x10010024	**Z[7]**	**0xfffff308**	**0x10010088**	**Z[32]**	**0xfffff298**
0x10010028	**Z[8]**	**0xfffff2f8**	**0x1001008c**	**Z[33]**	**0xfffff298**
0x1001002c	**Z[9]**	**0xfffff2f8**	**0x10010090**	**Z[34]**	**0xfffff298**
0x10010030	**Z[10]**	**0xfffff2f8**	**0x10010094**	**Z[35]**	**0xfffff298**
0x10010034	**Z[11]**	**0xfffff2f8**	**0x10010098**	**Z[36]**	**0xfffff288**
0x10010038	**Z[12]**	**0xfffff2e8**	**0x1001009c**	**Z[37]**	**0xfffff288**
0x1001003c	**Z[13]**	**0xfffff2e8**	**0x100100a0**	**Z[38]**	**0xfffff288**
0x10010040	**Z[14]**	**0xfffff2e8**	**0x100100a4**	**Z[39]**	**0xfffff288**
0x10010044	**Z[15]**	**0xfffff2e8**	**0x100100a8**	**Z[40]**	**0xfffff278**
0x10010048	**Z[16]**	**0xfffff2d8**	**0x100100ac**	**Z[41]**	**0xfffff278**
0x1001004c	**Z[17]**	**0xfffff2d8**	**0x100100b0**	**Z[42]**	**0xfffff278**
0x10010050	**Z[18]**	**0xfffff2d8**	**0x100100b4**	**Z[43]**	**0xfffff278**
0x10010054	**Z[19]**	**0xfffff2d8**	**0x100100b8**	**Z[44]**	**0xfffff268**
0x10010058	**Z[20]**	**0xfffff2c8**	**0x100100bc**	**Z[45]**	**0xfffff268**
0x1001005c	**Z[21]**	**0xfffff2c8**	**0x100100c0**	**Z[46]**	**0xfffff268**
0x10010060	**Z[22]**	**0xfffff2c8**	**0x100100c4**	**Z[47]**	**0xfffff268**
0x10010064	**Z[23]**	**0xfffff2c8**	**0x100100c8**	**Z[48]**	**0xfffff258**
0x10010068	**Z[24]**	**0xfffff2b8**	**0x100100cc**	**Z[49]**	**0xfffff258**

运行结果显示



代码段内存映像

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x3c011001	lui $1, 0x00001001	8: lw $s1, K # $s1代表K
	0x00400004	0x8c310000	lw $17, 0x00000000($1)	
	0x00400008	0x3c011001	lui $1, 0x00001001	9: lw $s2, Y # $s2代表Y
	0x0040000c	0x8c320004	lw $18, 0x00000004($1)	
	0x00400010	0x3c011001	lui $1, 0x00001001	10: la $s3, Z # $s3代表数组Z
	0x00400014	0x34330008	ori $19, $1, 0x00000008	
	0x00400018	0x2a280032	slti $8, $17, 0x00000032	12: slti $t0, $s1, 50 # 若K<50，$t0 = 1，否则$t0 = 0
	0x0040001c	0x1100000f	beq $8, $0, 0x0000000f	13: beq $t0, $0, exit # 若$t0 = 0，即K>=50，则跳转到exit（跳出循环）
	0x00400020	0x00114882	srl $9, $17, 0x00000002	14: srl $t1, $s1, 2 # 将$s1右移两位，即$t1 = K / 4
	0x00400024	0x212900d2	addi $9, $9, 0x000000d2	15: addi $t1, $t1, 210 # $t1 = $t1 + 210
	0x00400028	0x00094900	sll $9, $9, 0x00000004	16: sll $t1, $t1, 4 # 将$t1左移4位，即$t1 = $t1 * 16
	0x0040002c	0x02494822	sub $9, $18, $9	17: sub $t1, $s2, $t1 # $t1 = $s2(Y) - $t1
	0x00400030	0xae690000	sw $9, 0x00000000($19)	18: sw $t1, 0($s3) # 将$t1的值存入$s3(数组Z)的相应位置中
	0x00400034	0x8e640000	lw $4, 0x00000000($19)	19: lw $a0, 0($s3) # 输出数组Z当前位置的值
	0x00400038	0x24020001	addiu $2, $0, 0x00000001	20: li $v0, 1
	0x0040003c	0x0000000c	syscall	21: syscall
	0x00400040	0x3c011001	lui $1, 0x00001001	22: la $a0, str # 输出空格“ ”
	0x00400044	0x342400d0	ori $4, $1, 0x000000d0	
	0x00400048	0x24020004	addiu $2, $0, 0x00000004	23: li $v0, 4
	0x0040004c	0x0000000c	syscall	24: syscall
	0x00400050	0x22730004	addi $19, $19, 0x00000004	25: addi $s3, $s3, 4 # 让数组Z寻址到下一个位置（4个字节）
	0x00400054	0x22310001	addi $17, $17, 0x00000001	26: addi $s1, $s1, 1 # K = K + 1
	0x00400058	0x08100006	j 0x00400018	27: j loop # 跳转到loop，继续循环
	0x0040005c	0x2402000a	addiu $2, $0, 0x0000000a	29: li $v0, 10 # 程序终止结束
	0x00400060	0x0000000c	syscall	30: syscall

地址　　机器码　　汇编指令

[00400014]　0c100009　**jal 0x00400024 [main]**

[00400018]　00000000　**nop**

[0040001c]　3402000a　**ori $2, $0, 10**

[00400020]　0000000c　**syscall**

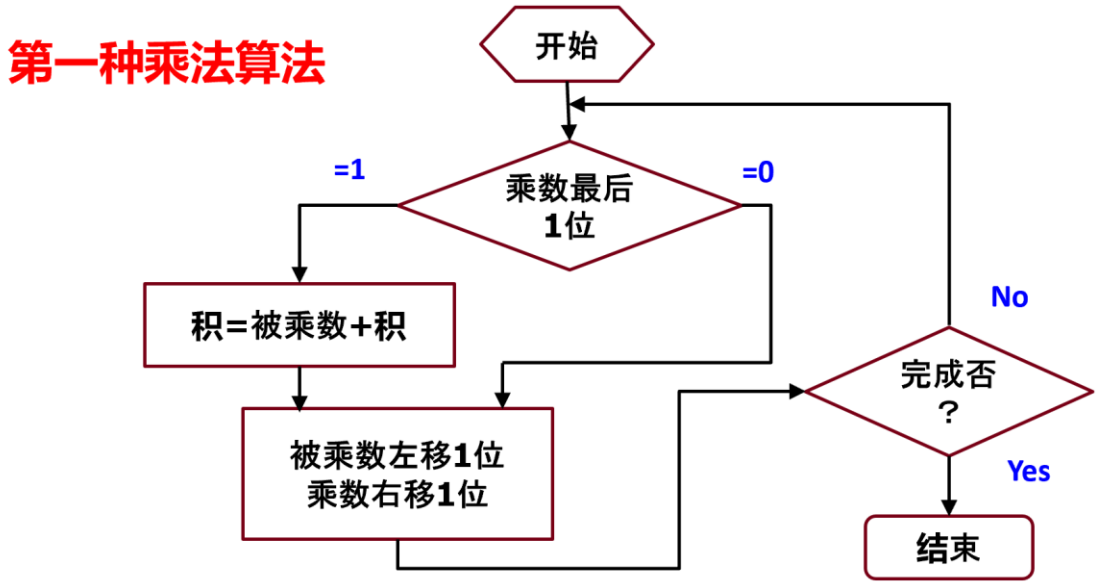
[00400024]　3c101001　**lui $16, 4097 [z]**

…

实验 2. 熟悉无符号乘法操作，并写出汇编代码 multu

#　乘数放在 $a0 和 $a1

#　结果放在 $v0 核 $v1 中



程序设计思路：

两个 32 位寄存器相乘时，如果得到的积 32 位寄存器不够放（溢出），就要用高 32 位寄存器存储，即用两个 32 位寄存器拼成一个 64 位寄存器存储积。

当 积的低 32 位 = 积的低 32 位 + 被乘数的低 32 位 的结果反而变小时，说明发生溢出，要向积的高 32 位进 1。再用积的高 32 位加上被乘数的高 32 位即可得到正确的积高 32 位。其他步骤按上面的流程图，重复循环 32 次，即可得到正确的结果。

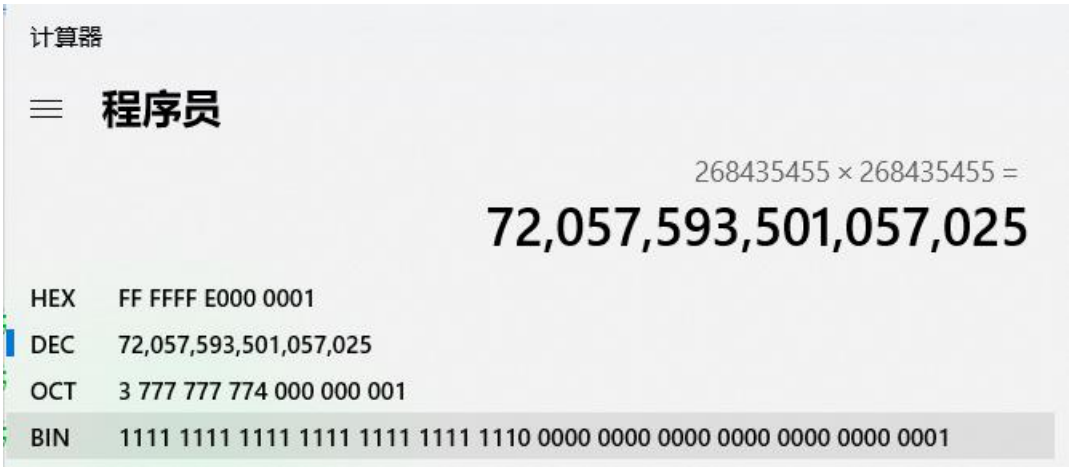
代码如下图：

```
1         .data
2   X:    .word   268435455
3   Y:    .word   268435455
4         .text
5   main:
6         lw      $a0, X              # $a0代表被乘数的低32位
7         lw      $a1, Y              # $a1代表乘数
8         addi    $a2, $0, 0          # $a2代表被乘数的高32位
9         addi    $v0, $0, 0          # $v0代表积的高32位
10        addi    $v1, $0, 0          # $v1代表积的低32位
11        addi    $s0, $0, 32         # $s0代表循环次数
12  loop:
13        andi    $t0, $a1, 1         # 如果乘数最低位是1，$t0 = 1
14        beq     $t0, $0, then       # 乘数最低位是0，跳转到then
15                                    # 乘数最低位是1，那么...
16        addu    $v1, $a0, $v1       # 积的低32位 = 被乘数低32位 + 积的低32位
17        sltu    $t0, $v1, $a0       # 如果积的低32位 < 被乘数低32位，说明发生进位，要向高位进1
18        addu    $v0, $v0, $t0       # 积的高32位 = 积的高32位 + 进位
19        addu    $v0, $v0, $a2       # 积的高32位 = 积的高32位 + 被乘数的高32位
20  then:
21        srl     $a1, $a1, 1          # 乘数右移一位
22        sll     $a2, $a2, 1          # 被乘数高32位左移一位
23        andi    $t0, $a0, 0x80000000 # 如果被乘数低32位的最高位是1，$t0 = 1
24        sll     $a0, $a0, 1          # 被乘数低32位左移一位
25        beq     $t0, $0, L           # 如果被乘数低32位的最高位是0，即$t0 = 0,跳转到L
26        ori     $a2, $a2, 1          # 如果被乘数低32位的最高位是1，被乘数高32位的最低位 置1
27  L:
28        addi    $s0, $s0, -1         # 循环次数减1
29        beq     $s0, $0, final       # 如果循环32次，跳转到final
30        j       loop                 # 未循环32次，跳转到loop，继续循环
31  final:
32        addu    $a0, $v0, $0         # 输出积的高32位
33        li      $v0, 35              # 输出二进制
34        syscall
35        addu    $a0, $v1, $0         # 输出积的低32位
36        li      $v0, 35              # 输出二进制
37        syscall
38        li      $v0, 10              # 程序结束
39        syscall
```

程序输出结果如下图：

```
00000000111111111111111111111111100000000000000000000000000000001
-- program is finished running --
```

对比计算器结果发现程序结果正确！



思考：如何扩展为带符号的乘法运算，并写出代码。

程序设计思路：

对于有符号的乘法运算，可以先通过被乘数与乘数的最高位是否是 1 判断出它们的正负。如果被乘数与乘数符号相异，那么最后的结果也是负的。接着，对负的被乘数或乘数取它们的补码，得到正的被乘数或乘数。 然后就可使用之前的无符号乘法运算得到积。但如果最后结果是负的话，只要把得到的积取补码便可得到正确的结果。

代码如下：

```
        .data
X:      .word   -1073741824
Y:      .word   1073741824
        .text
main:
        lw      $a0, X                  # $a0代表被乘数的低32位
        lw      $a1, Y                  # $a1代表乘数
        addi    $a2, $0, 0              # $a2代表被乘数的高32位
        addi    $v0, $0, 0              # $v0代表积的高32位
        addi    $v1, $0, 0              # $v1代表积的低32位
        addi    $s0, $0, 32             # $s0代表循环次数

        andi    $t0, $a0, 0x80000000    # 如果被乘数最高位是1，说明它是负数，$t0 = 1
        andi    $t1, $a1, 0x80000000    # 如果乘数最高位是1，说明它是负数，$t1 = 1
        xor     $t2, $t0, $t1           # 如果被乘数和乘数符号相异，那么积是负数，$t2 = 1
j1:     bne     $t0, $0, neg1           # 如果被乘数是负数，跳转到neg1，取它的补码
j2:     bne     $t1, $0, neg2           # 如果乘数是负数，跳转到neg2，取它的补码
        j       loop                    # 跳转到loop，开始计算乘积

```

```
neg1:   xori    $a0, $a0, 0xFFFFFFFF        # 将被乘数取反
        addi    $a0, $a0, 1                 # 得到被乘数的补码
        j       j2                          # 跳转到j2，继续判断乘数是否是负数
neg2:   xori    $a1, $a1, 0xFFFFFFFF        # 将乘数取反
        addi    $a1, $a1, 1                 # 得到乘数的补码
        j       loop                        # 跳转到loop，开始计算乘积

loop:
        andi    $t0, $a1, 1                 # 如果乘数最低位是1，$t0 = 1
        beq     $t0, $0, then               # 乘数最低位是0，跳转到then
                                            # 乘数最低位是1，那么...
        addu    $v1, $a0, $v1               # 积的低32位 = 被乘数低32位 + 积的低32位
        sltu    $t0, $v1, $a0               # 如果积的低32位 < 被乘数低32位，说明发生进位，要向高位进1
        addu    $v0, $v0, $t0               # 积的高32位 = 积的高32位 + 进位
        addu    $v0, $v0, $a2               # 积的高32位 = 积的高32位 + 被乘数的高32位
then:
        srl     $a1, $a1, 1                 # 乘数右移一位
        sll     $a2, $a2, 1                 # 被乘数高32位左移一位
        andi    $t0, $a0, 0x80000000        # 如果被乘数低32位的最高位是1，$t0 = 1
        sll     $a0, $a0, 1                 # 被乘数低32位左移一位
        beq     $t0, $0, L                  # 如果被乘数低32位的最高位是0，即$t0 = 0,跳转到L
        ori     $a2, $a2, 1                 # 如果被乘数低32位的最高位是1，被乘数高32位的最低位 置1
```

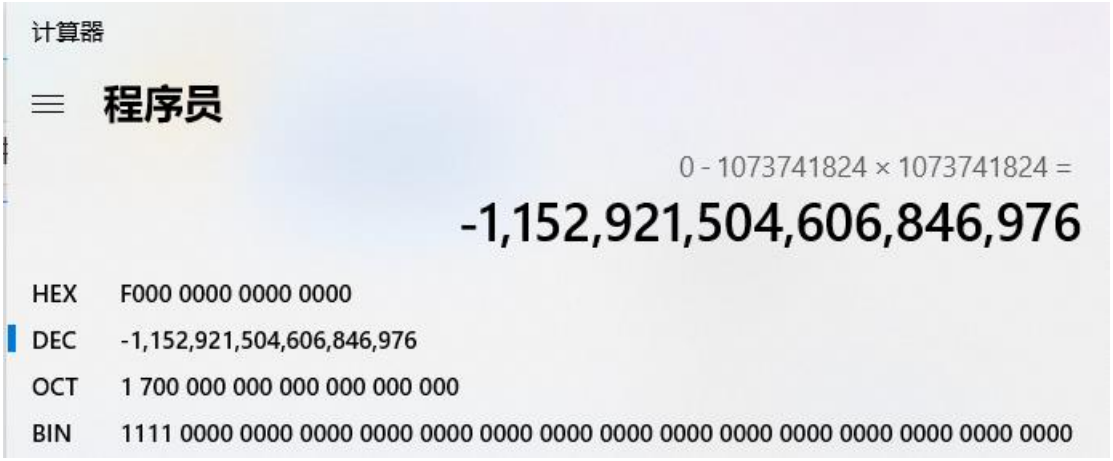
```
L:
        addi    $s0, $s0, -1                # 循环次数减1
        beq     $s0, $0, final              # 如果循环32次，跳转到final
        j       loop                        # 未循环32次，跳转到loop，继续循环

final:
        bne     $t2, $0, neg3               # 如果最后结果是负的，跳转到neg3求整个积的补码
print:
        addu    $a0, $v0, $0                # 输出积的高32位
        li      $v0, 35                     # 输出二进制
        syscall
        addu    $a0, $v1, $0                # 输出积的低32位
        li      $v0, 35                     # 输出二进制
        syscall
        li      $v0, 10                     # 程序结束
        syscall
```

```
neg3:   xori    $v0, $v0, 0xFFFFFFFF        # 将积的高32位取反
        xori    $v1, $v1, 0xFFFFFFFF        # 将积的低32位取反
        beq     $v1, 0xFFFFFFFF, P          # 如果积的低32位是0xFFFFFFFF,加1会溢出，所以跳转到P进行防溢出操作
        addi    $v1, $v1, 1                 # 积的低32位加1（取补码）
        j       print                       # 跳转到print
P:      addi    $v1, $0, 0                  # 积的低32位清零
        addi    $v0, $v0, 1                 # 积的高32位加1
        j       print                       # 跳转到print
```

程序调试与输出：

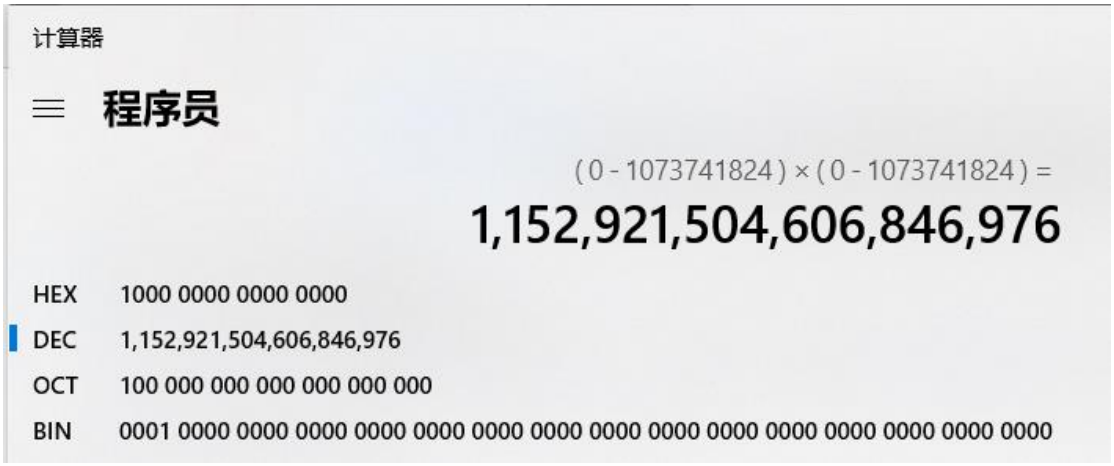
-1073741824 * 1073741824

```
11110000000000000000000000000000000000000000000000000000000000000000
— program is finished running —
```



（-1073741824）* （-1073741824）

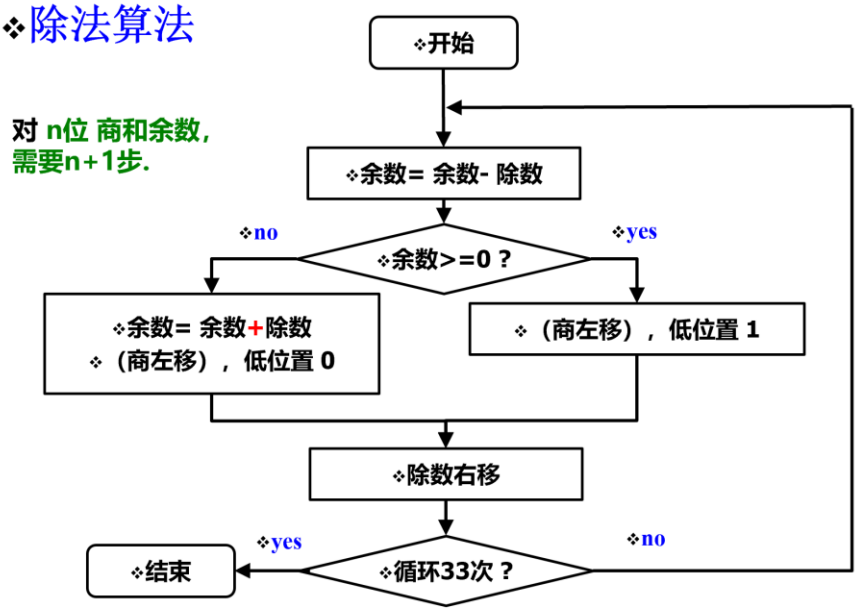
```
0001000000000000000000000000000000000000000000000000000000000000
— program is finished running —
```



实验 3. 熟悉无符号乘法操作，并写出汇编代码 divu

```
#    被除数放在$a0 中，除数放在$a1 中
#    商和余数分别放在$v0  核  $v1
```

❖除法算法



设计思路：

先将除数加载到高 32 位寄存器，不断右移循环，如果高 32 位寄存器最低位是 1，就把 1 加载到低 32 位寄存器的最高位。当除数全部加载到低 32 位寄存器且没有溢出，高 32 位寄存器为 0 时，即可进行正常的无符号除法运算，如上流程图所示。这样，一共进行 33 次循环，一定能得到正确的商和余数。

代码如下图：

```
        .data
X:      .word   20000000
Y:      .word   8000000
        .text
main:
        lw      $a0,X                   # $a0为余数
        lw      $a1,Y                   # $a1为除数的高32位
        addi    $a2,$0,0                # $a2为除数的低32位
        addi    $s0,$0,0xFFFFFFFE       # $s0用于最低位 置0
        addi    $s1,$0,1                # $s1用于最低位 置1，并可判断最低位是否是1
        addi    $s2,$0,0x80000000       # $s2用于最高位 置1
        addi    $s3,$0,33               # $s3用于判断循环次数

loop:                                   # 判断高位是0，低位大于0，可进行正常无符号除法
        sgt     $t2,$a1,$0              # 如果除数高32位大于0，$t2 = 1
        slti    $t3,$a2,1               # 如果除数低32位小于1，$t3 = 1
        or      $t4,$t2,$t3             # 满足以上两个条件之一，就不能进行正常无符号除法运算
        beq     $t4,$0,func             # 如果除数高32位为0，低32位大于0，可进行正常无符号除法运算，跳转到func

```

```
20          and     $t1, $a1, $s1           # 如果高32位寄存器除数的最低位是1，$t1 = 1
21          beq     $t1, $0, then1          # 如果$t1等于0，跳转到then1
22                                          # 如果高32位寄存器除数的最低位是1
23          srl     $a2, $a2, 1             # 将除数低32位右移一位
24          or      $a2, $a2, $s2           # 将除数低32位的最高位 置1
25          srl     $a1, $a1, 1             # 将除数高32位右移一位
26          j       then2                   # 跳转到then2（此时已完成除数右移，判断次数即可）
27
28  func:                                   # 正常的无符号除法运算
29          sub     $a0, $a0, $a2           # 余数 = 余数 - 除数
30          slti    $t0, $a0, 0
31          beq     $t0, $0, L              # 如果余数大于0，跳转到 L
32
33                                          # 余数小于0，那么...
34          add     $a0, $a0, $a2           # 余数 = 余数 + 除数
35          sll     $v0, $v0, 1             # 将商左移一位
36          and     $v0, $v0, $s0           # 将商的最低位 置0
37          j       then1                   # 跳转到then1
38
```

```
39  L:                                      # 余数大于0，那么...
40          sll     $v0, $v0, 1             # 将商左移一位
41          or      $v0, $v0, $s1           # 将商的最低位 置1
42          j       then1                   # 跳转到then1
43
44  then1:  srl     $a2, $a2, 1             # 除数低32位右移一位
45          srl     $a1, $a1, 1             # 除数高32位右移一位
46  then2:  addi    $s3, $s3, -1            # 次数减1
47          beq     $s3, $0, final          # 如果循环33次，跳转到final
48          j       loop                    # 未循环33次，继续循环
```

```
49  final:
50          add     $v1, $a0, $0            # 将余数保存到$v1寄存器
51          add     $a0, $v0, $0            # 输出商
52          li      $v0, 1
53          syscall
54          li      $v0, 11                 # 输出回车
55          li      $a0, '\n'
56          syscall
57          add     $a0, $v1, $0            # 输出余数
58          li      $v0, 1
59          syscall
60          li      $v0, 10                 # 程序结束
61          syscall
```

程序输出结果如下图：

2000000 / 8000000 = 2 …… 4000000

```
2
4000000
-- program is finished running --
```

要求：每一位同学完成实验报告，附上程序设计思路，实验结果截图