

实验 5：汇编语言编程练习 2

22336313 郑鸿鑫

一． 实验目的:

熟悉 MIPS 汇编程序开发环境，学习使用 MARS 工具，知道如何查看内存分配，熟悉无符号乘法操作

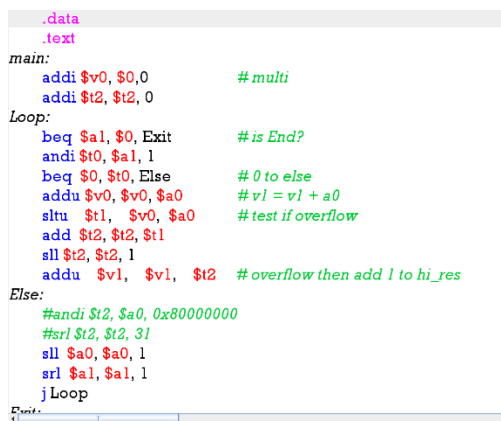
二． 实验内容和过程:

实验 1：汇编语言实现以下伪代码，使用移位指令实现乘除法运算

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。

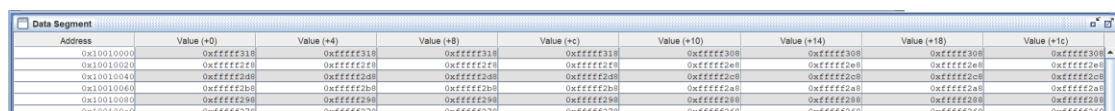
```
Int main ()
{
    Int K,Y;
    Int Z[50];
    Y=56;
    For(k=0;k<50;K++) Z[k]=Y-16*(k/4+210);
}
```

代码截图：



```
.data
.text
main:
    addi $v0, $0, 0      # multi
    addi $t2, $t2, 0
Loop:
    beq $a1, $0, Exit    # is End?
    andi $t0, $a1, 1
    beq $0, $t0, Else    # 0 to else
    addu $v0, $v0, $a0    # v1 = v1 + a0
    sltu $t1, $v0, $a0    # test if overflow
    add $t2, $t2, $t1
    sll $t2, $t2, 1
    addu $v1, $v1, $t2    # overflow then add 1 to hi_res
Else:
    #andi $t2, $a0, 0x80000000
    #srl $t2, $t2, 31
    sll $a0, $a0, 1
    srl $a1, $a1, 1
    j Loop
Exit:
```

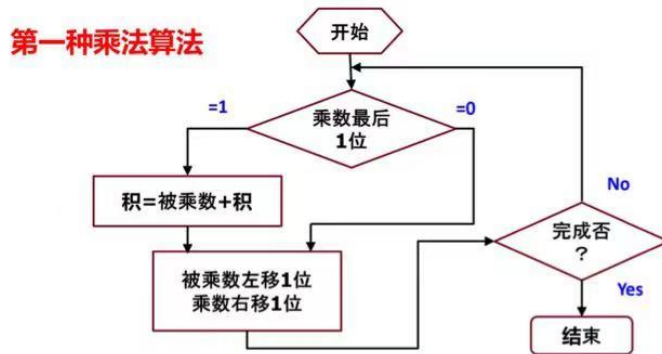
运行结果展示：



Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0xffffffff	0xffffffff	0xffffffff	0xffffffff	0xffffffff	0xffffffff	0xffffffff	0xffffffff
0x10010004	0xffffffff	0xffffffff	0xffffffff	0xffffffff	0xffffffff	0xffffffff	0xffffffff	0xffffffff
0x10010008	0xffffffff	0xffffffff	0xffffffff	0xffffffff	0xffffffff	0xffffffff	0xffffffff	0xffffffff
0x1001000c	0xffffffff	0xffffffff	0xffffffff	0xffffffff	0xffffffff	0xffffffff	0xffffffff	0xffffffff
0x10010010	0xffffffff	0xffffffff	0xffffffff	0xffffffff	0xffffffff	0xffffffff	0xffffffff	0xffffffff
0x10010014	0xffffffff	0xffffffff	0xffffffff	0xffffffff	0xffffffff	0xffffffff	0xffffffff	0xffffffff

实验 2：将乘数放在\$a0,\$a1 中，将结果放在\$v0,\$v1 中

第一种乘法算法



思考：如何扩展为带符号的乘法运算，并写出代码。

代码截图：

```

Multi:
subi $sp, $sp, 8
sw $a0, 0($sp)      # save $a0 in $sp
sw $a1, 4($sp)      # save $a1 in $sp+4
li $v0, 0
li $v1, 0
li $t0, 0

Loop:
andi $t1, $a1, 1    # test if the lowest bit of B is 0
beq $t1, 0, shift   # if true then don't add A
addu $v0, $a0, $v0  # else add A to res
addu $v1, $t3, $v1
sltu $t1, $v0, $a0  # test if overflow
addu $v1, $v1, $t1  # overflow then add 1 to hi_res

shift
andi $t1, $a0, 0x80000000 # test if the highest bit of lo_A is 1
srl $t1, $t1, 31         # adjust result to 1
sll $a0, $a0, 1         # shift A
sll $t3, $t3, 1
addu $t3, $t3, $t1      # adjust A
srl $a1, $a1, 1         # shift B
addiu $t0, $t0, 1       # ++i
sltiu $t1, $t0, 32      # loop if i < 32
bne $t1, $zero, Loop
lwr $a0, 0($sp)         # load $a0 in $sp
lwr $a1, 4($sp)         # load $a1 in $sp+4
  
```

运行结果展示（以 $5 \times 6 = 30$ 为例）：

结果的高 32 位存储在 \$v0 中，低 32 位存储在 \$v1 中

Step	Address	Code	Basic	Source
0	0x00400000	0x00100000	addi \$1, \$0, 0x00000000	2: subi \$sp, \$sp, 8
1	0x00400004	0x03a1e822	sub \$29, \$29, \$1	
2	0x00400008	0xafa80000	sw \$a0, 0(\$sp)	3: sw \$a0, 0(\$sp)
3	0x0040000c	0xafa80004	sw \$a1, 4(\$sp)	4: sw \$a1, 4(\$sp)
4	0x00400010	0x24020000	addiu \$2, \$0, 0x00000000	8: li \$v0, 0
5	0x00400014	0x24030000	addiu \$3, \$0, 0x00000000	9: li \$v1, 0
6	0x00400018	0x24030000	addiu \$4, \$0, 0x00000000	10: li \$v0, 0
7	0x0040001c	0x24110001	addiu \$17, \$0, 0x00000000	11: li \$t1, 0x00000001
8	0x00400020	0x30380000	sll \$1, 0xffff0000	12: li \$a2, 0x00000000
9	0x00400024	0x42020000	addi \$19, \$1, 0x00000000	
10	0x00400028	0x02284024	and \$v0, \$a1, \$a1	15: and \$v0, \$a1, \$a1
11	0x0040002c	0x11000000	beq \$v0, \$zero, 13	16: beq \$v0, \$zero, 13
12	0x00400030	0x00050102	addu \$v0, \$v0, \$v0	18: addu \$v0, \$v0, \$v0
13	0x00400034	0x00418212	addu \$v1, \$v1, \$a0	19: addu \$v1, \$v1, \$a0
14	0x00400038	0x00648200	sltu \$v1, \$v1, \$a0	20: sltu \$v1, \$v1, \$a0
15	0x0040003c	0x00491021	addu \$v0, \$v0, \$t1	21: addu \$v0, \$v0, \$t1

有符号乘法代码：

Multi:

```

subi $sp, $sp, 8

sw $a0, 0($sp)      # save $a0 in $sp
sw $a1, 4($sp)      # save $a1 in $sp+4

li $v0, 0
li $v1, 0
li $t0, 0

andi $t1, $a0, 0x80000000 # signed extend A
beqz $t1, Loop
  
```

```

    not    $t3,    $t3
Loop:
    andi   $t1,    $a1,    1        # test if the lowest bit of B is 0
    beq    $t1,    0,      shift    # if true then don't add A
    addu   $v0,    $a0,    $v0      # else add A to res
    addu   $v1,    $t3,    $v1
    sltu   $t1,    $v0,    $a0      # test if overflow
    addu   $v1,    $v1,    $t1      # overflow then add 1 to hi_res
shift:
    andi   $t1,    $a0,    0x80000000 # test if the highest bit of lo_A is 1
    srl    $t1,    $t1,    31        # adjust result to 1
    sll    $a0,    $a0,    1        # shift A
    sll    $t3,    $t3,    1
    addu   $t3,    $t3,    $t1      # adjust A
    srl    $a1,    $a1,    1        # shift B
    addiu   $t0,    $t0,    1        # ++i
    sltiu   $t1,    $t0,    31      # loop if i < 31
    bne    $t1,    $zero,    Loop
    seq    $t1,    $t0,    32      # corner case: if B is signed then res -= A, which means A
= -A and loop it once again
    bne    $t1,    $zero,    Done
    andiu   $t1,    $a1,    1        # test if is signed
    beq    $t1,    $zero,    Done    # if not then done
    not     $a0,    $a0              # else A = -A and loop
    not     $t3,    $t3
    addiu   $a0,    $a0,    1        # not and plus 1
    seq     $t1,    $a0,    0        # if lo_A is 0 then overflow
    addu   $v1,    $v1,    $t1      # overflow then add 1 to hi_res
    j      Loop
Done:
    lw      $a0,    0($sp)
    lw      $a1,    4($sp)
    addiu   $sp,    $sp,    8
    jr      $ra

```

运行结果展示：

\$v0	2	0xffffffffc
\$v1	3	0x00000048
\$a0	4	0x00000008
\$a1	5	0x80000009

大数据测试样例：（有符号和无符号）

```

Multu: 16564564 * 15434575 = 255667005400300
00000000000000000000111010001000011100011101000010011010010011101100
-- program is finished running --
result: 0b00000000000000000000111010001000011100011101000010011010010011101100 = 255667005400300
Mult: -423223 * 85434239 = -36157734932297
111111111111111111101111000111010101111000011011011000010110111
-- program is finished running --

```

实验 3：实现 swap 代码，作用是将两个寄存器中的数交换，而且必须使用栈来保存局部变量。

代码截图：

```

.data
num1: .word 1
num2: .word 2
.text
la $a0, num1
la $a1, num2
swap:
    addi $sp, $sp, -4
    lw $t0, 0($a0) # temp = *px
    sw $t0, 0($sp)
    lw $t0, 0($a1) # *px = *py
    sw $t0, 0($a0)
    lw $t0, 0($sp) # *py = temp
    sw $t0, 0($a1)
    addi $sp, $sp, 4
    lw $a0, 0($a0)
    li $v0, 1
    syscall
    lw $a0, 0($a1)
    li $v0, 1
    syscall

```

运行结果展示：

```

21
-- program is finished running (dropped off bottom) --
Clear

```

说明：代码中将 num1（1）和 num2（2）存入 \$a0 和 \$a1 中，经过汇编程序后，依次输出 \$a0 和 \$a1 中的数，得到结果为 21，说明交换成功。

三．实验总结

1. 实验中较难实现的应该是第二个实验，一开始没有考虑到 32 位数乘法的溢出问题，会导致乘法的最终结果最坏的情况下会达到 64 位，所以依照要求，将结果的高 32 位存储在 \$v0 中，低 32 位存储在 \$v1 中。基于无符号乘法改进为带符号乘法，只需要进行以下改动，在初始化 A 时对高 32 位进行符号扩展（即符号位为 1 则用 1 补齐，符号位为 0 则用 0 补齐）然后在最后一次循环中如果 B 符号位置位，则应将积减去 A,也就是加上 -A.
2. 无符号位移可以当作乘除 2 的幂运算，带符号算数也可以，对于负数右移，可以证明舍入方向为负方向。
3. 无符号加法和带符号加法在硬件运算上是一样的，区别只在于溢出时带符号运算会抛出异常，而无符号不会。因此在需要自然溢出时应使用无符号加法。
4. 栈的地址从高到低减小，函数开始时将栈指针下移，使用偏移寻址访问栈指针以上分配给该函数的内存空间，函数执行完毕在返回前将栈指针上移。