计算机体系架构　第一周第二次作业

范云潜 18373486

微电子学院 184111 班

日期：2020 年 9 月 16 日

作业内容：3.2, 3.4, 3.6, 3.8;2.1, 2.2, 2.5, 2.6;

Problem 3.2

根据补码取负值的原则 `2047` = `0x000007ff` ， $-$`2047` = `0xfffff801` 。

Problem 3.4

表示为十六进制为 `0xffffff06` ，以补码换算到十进制 $-$`250` 。

Problem 3.6

表示为十六进制为 `0x7fffffef` ，以补码换算到十进制 `2147483631` 。

Problem 3.8

对于 Harry ，可以举反例 `0b1111` $*$ `0b0110` = `0b1011010` 。

对于 David ，可以举反例 `36 + 6 = 0b100100 + 0b110 = 0b101010` ，仅有 3 个‘1’。

Problem 2.1

原始代码

```
        addi    $v0, $zero, 0 # the increase is after init
        # I: 001000 00000 00010  0000000000000000
loop:   lw      $v1, 0($a1)
        # I: 100011 00101 00011  0000000000000000
        sw      $v1, 0($a1)
        # I: 000010 00101 00011  0000000000000000
        addi    $a0, $a0,   4
        # I: 001000 00100 00100  0000000000000100
        addi    $a1, $a1,   4
```

```
        # I: 001000 00101 00101  0000000000000100
        beq     $v1, $zero, loop # loop condition
        # I: 000100 00000 00011  1111111111111011 (-5 * 4) 0101 1011
```

按照汇编格式，写成指令 / 对应十进制数的形式。

```
0x20020000 / 537001984
0x8ca30000 / 2359492608
0xaca30000 / 2896363520
0x20840004 / 545521668
0x20a50004 / 547684356
0x1060fffb / 274792443
```

bug 修改后：

```
        addi    $v0, $zero, -1 # the increase is after init
loop:   lw      $v1, 0($a1)
        sw      $v1, 0($a1)
        addi    $a0, $a0,   4
        addi    $a1, $a1,   4
        addi    $v0, $v0,   1 # add 1 to the cnt
        bne     $v1, $zero, loop # loop condition
```

Problem 2.2

0x7ffffffa = 0b01111111111111111111111111111010 = 2147483642

Problem 2.5

```
# assign t0=a, s0=b, s1=c, s2=d
    .data
mask:
    .word 0xfffff83f
    .text
start:
    lw   $t0, mask  # a=0xfffff83f   1000 0011 1111
    lw   $s0, shifter #
    and  $s0, $s0, $t0      # apply the mask to b
    andi $s2, $s2, 0x1f     # select low-5 bit of d
    sll  $s2, $s2, 6
    or   $s0, $s0, $s2      # fill the masked bits of b
    sw   $s0, shifter       # change the shamt
shifter:
    sll  $s0, $s1, 0
```

整体流程如下：

1. 加载一个 MASK，可以通过与操作清空 R 型指令 `sll` 的 shamt 区段
2. 通过另一个 MASK 选择需要移位的位数，也就是 `$s2` 的低 5 位
3. 将这个位数加载到 shamt 区段再执行指令即可完成自定义位数的移位操作

但是我们可以认识到，这是一种不显式的操作，无法通过阅读单独的一条指令确定其功能，这样的写法使得指令存在耦合，增大了程序调试与维护的难度。

Problem 2.6

为了不出现位扩展，采用逻辑移位操作。

```
sll $t3, $t3, 10
srl $t1, $t3, 15
```

词汇

explicit 显式的