计算机组成原理实验报告

姓名： 张佳晨　　学号：11713020

一、 实验目的

1. Learn how to exchange the highest 8 bits with the lowest 8 bits in a word.
2. Know about calculating the bit inversion (0->1,1->0) of the odd digits in a word.
3. Learn about logically shifting. For example, given an integer x, calculate the result of 10x. DO NOT use mult/mul/multu in your code.
4. Calculate the absolute value of a word by basic operations other than abs.

二、 实验内容

1. Exchange the highest 8 bits with the lowest 8 bits in a word.
2. Calculate the bit inversion (0->1,1->0) of the odd digits in a word.
3. For an integer x, calculate the result of 10x. DO NOT use mult/mul/multu in your code.
4. Calculate the absolute value of a word by basic operations other than abs.

三、 实验步骤（阐述代码思路或操作步骤）

Problem 1

> Save the variable in a word so that I can use `lb` and `sb`, as well as indexed addressing to exchange the highest and lowest 8 bits in this word.

```
.macro print_string(%str)
    .data
    pstr: .asciiz %str
    .text
    la $a0,pstr
    li $v0,4
    syscall
.end_macro

.macro END
    li $v0,10
    syscall
.end_macro
# text segment
    .text
    .global main
main:                       # execution starts here

    print_string("Please input a integer that you want to shift the lowest 8-bit and the
highest 8-bit: ")
```

```
    li $v0, 5
    syscall                 # system call to read integer and store in $v0
    sw      $v0, value

    jal     shift               # jump to shift and save position to $ra

    print_string("After shift the lowest 8-bit and the highest 8-bit, the integer is : ")
    lw      $a0, value
    li $v0, 1               # system call to print integer
    syscall

    li $v0, 10              # system call to exit
    syscall

shift: # shift the lowest 8-bit and the highest 8-bit stored in value
    la  $t0, value
    lb  $t1, 0($t0)     # load the lowest 8-bits to $t1
    lb  $t2, 3($t0)     # load the highst 8-bits to $t2
    sb  $t2, 0($t0)     # store $t2 to the lowest 8-bits
    sb  $t1, 3($t0)     # store $t1 to the highest 8-bits
    jr      $ra                 # jump to $ra

    .data
value:  .word   0
```

Problem 2

> According to the truth table, we can inverse 0s and 1s by doing `xor` with 1.
>
> I am required to calculate the bit inversion of the odd bits, so I choose to do the `xor` with `01010101010101010101010101010101`, which is a 32-bit binary number.
>
> In hexadecimal number, it is `0x55555555`.

```
.macro print_string(%str)
    .data
    pstr: .asciiz %str
    .text
    la $a0,pstr
    li $v0,4
    syscall
.end_macro

.macro END
    li $v0,10
    syscall
.end_macro

# text segment
    .text
    .global main
main:                           # execution starts here
```

```
    print_string("Please input an integer x that you want to make odd_bit_inversion on: ")

    li $v0, 5
    syscall                     # system call to read integer and store in $v0
    move    $t0, $v0            # $t0 = $v0

    jal     odd_bit_inversion   # jump to odd_bit_inversion and save position to $ra

    print_string("After odd_bit_inversion, the value is: ")

    move $a0, $t0
    li $v0, 1                   # system call to print integer
    syscall

    END

odd_bit_inversion:              # make odd_bit_inversion for the value stored in $t0, and
store the result in  $t0
    xori    $t0, $t0, 0xaaaaaaaa         # 10101010101010101010101010101010
    jr      $ra                          # jump to $ra


# 11111111111111111111111111111111
# 00000000000000000000000000000000

# xor
# 0 0 0
# 1 0 1
# 0 1 1
# 1 1 0
```

Problem 3

> I can use bit shift operation to enlarge the number by factor 2.
>
> I am required to get the 10 times of the given number without using multiple operation.
>
> So first get 8 times of the number (shift left 3-bit logically) and then get 2 times of the number (shift left 1-bit logically) and get the sum of them, which is what I need.

```
.macro print_string(%str)
    .data
    pstr: .asciiz %str
    .text
    la $a0,pstr
    li $v0,4
    syscall
.end_macro

.macro END
    li $v0,10
    syscall
.end_macro
```

```
# text segment
    .text
    .global main
main:                       # execution starts here
    print_string("Please input an integer x that you want to get ten times of it >")
    li $v0, 5
    syscall                 # system call to read integer and store in $v0

    move    $a0, $v0        # $a0 = $v0
    jal     ten_times       # jump to ten_times and save position to $ra
    print_string("Ten times of your input is: ")

    move $a0, $t0
    li $v0, 1               # system call to print integer at $a0
    syscall

    END

ten_times:
    sll $t0, $a0, 3         # $t0 = 8 * $a0
    sll $a0, $a0, 1         # $a0 = 2 * $a0
    add $t0, $t0, $a0       # $v0 = $t0 + $a0
    jr      $ra             # jump to $ra
```

Problem 4

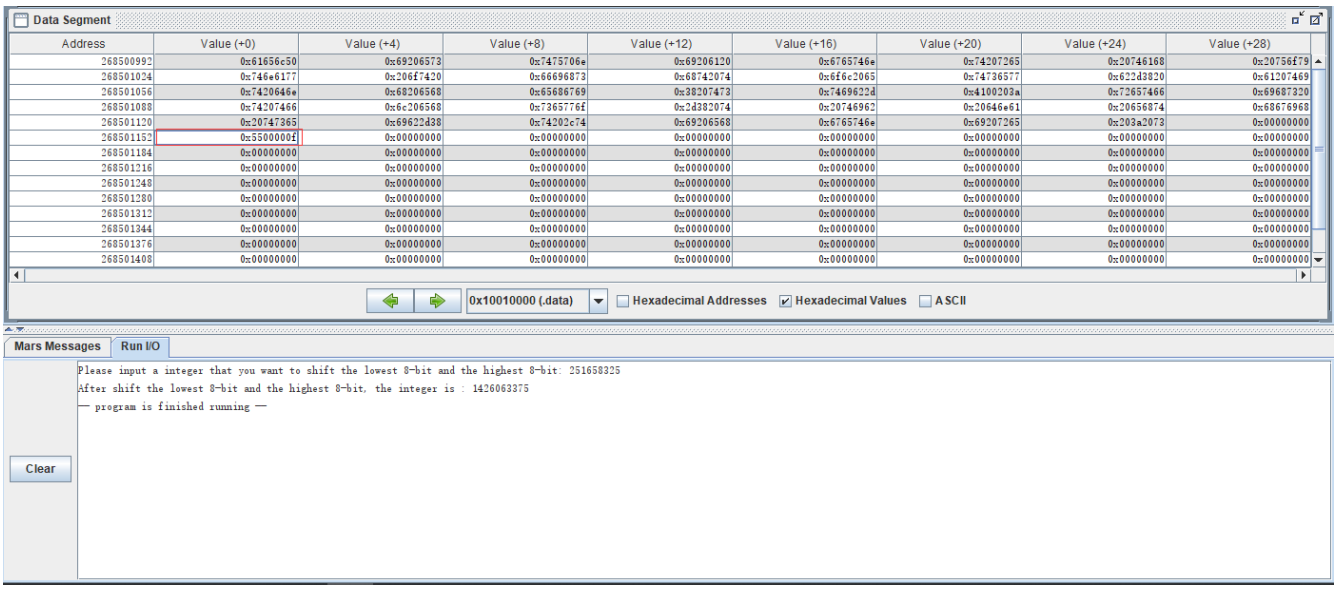
> (For signed binary numbers) As the negative numbers start with 1 and positive numbers start with 0, I first get the result of the number shifting right 31-bit logically and I can know whether it is a negative number or a positive number by checking the value.
>
> If it is a positive number, then I need to do nothing except returning it.
>
> If it is a negative number, as I learned before, I can get the opposite number by getting the 1's complement of the negative number and add 1 to it.

```
.macro print_string(%str)
    .data
    pstr: .asciiz %str
    .text
    la $a0,pstr
    li $v0,4
    syscall
.end_macro

.macro END
    li $v0,10
    syscall
.end_macro

# text segment
    .text
    .global main
```

```
main:                       # execution starts here
    print_string("Please input an integer x that you want to get absolute value of it >")
    li $v0, 5
    syscall                 # system call to read integer and store in $v0

    move    $t0, $v0        # $t0 = $v0
    jal     absolute        # jump to ten_times and save position to $ra
    print_string("The absolute value of your input is: ")

    move $a0, $t0
    li $v0, 1               # system call to print integer at $a0
    syscall

    END

absolute: # get absolute value of which store in $t0, and store the result in  $t0
    move    $t1, $t0        # $t1 = $t0
    srl     $t1, $t1, 31
    bne     $t1, $zero, negative    # if $t1 != $zero then negative
    jr      $ra

    negative:
    xori $t0, $t0, 0xffffffff   # 11111111111111111111111111111111 to make the signed-bit
always be 0
    addi    $t0, $t0 1          # $t0 = $t0 + 1
    jr      $ra                 # jump to $ra
```

四、 实验结果（截图并配以适当的文字说明）

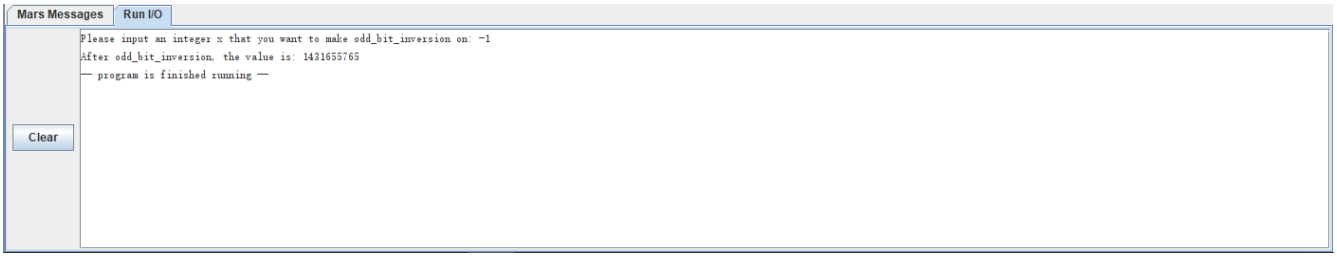
Problem 1



> Here I initially set the value is `251658325` = `0x0f000055`

> The output is `1426063375` = `0x5500000f`, which is shown in the picture.

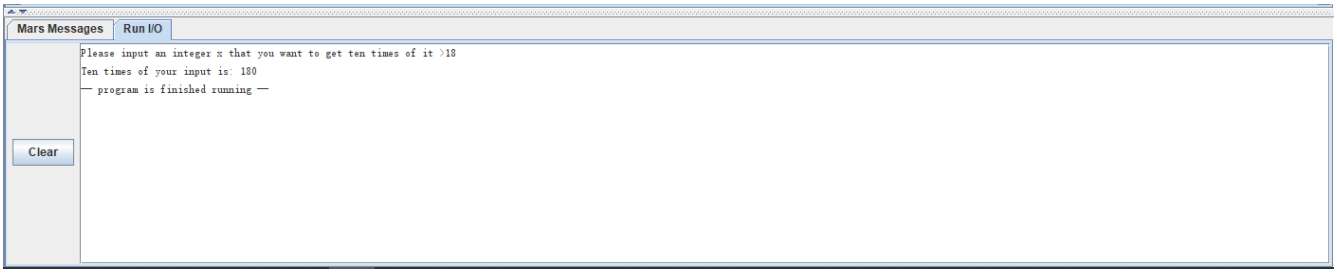
Problem 2



> Here I initially set the value is `-1` = `0xffffffff`

> The output is `1431655765` = `0x55555555`.

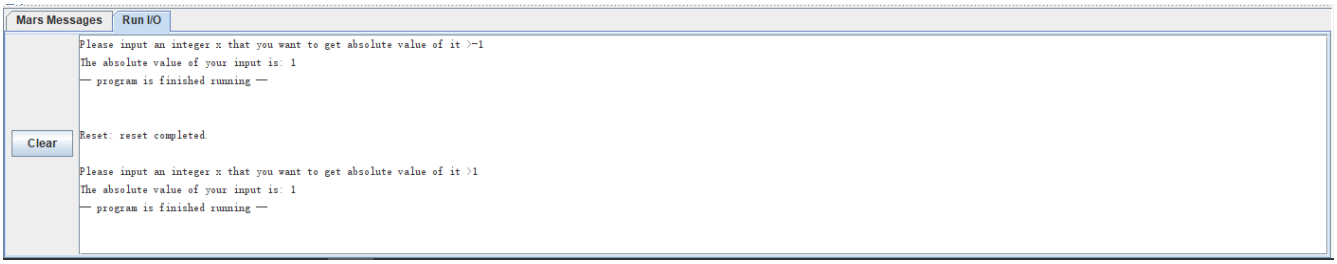
Problem 3



> Here I initially set the value is `18`

> The output is `180`

Problem 4



> Here I initially set the value is `-1`

> The output is `1`

> Then I set the value is `1`

> The output is `1`

五、 实验分析（遇到的问题以及解决方案）

> Not yet

六、 实验小结与体会

1. Shift left logically is an efficient way to do specific multiplication.
2. The comments is very important for writing assembly codes as one simple instruction (for high-level language) is constructed by many assembly instruction, because of the data accessing.
3. Using bit operation is an efficient way for programming.