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 about the overflow of the basic arithmetic (`add` and `mulitiply`).
2. Learn about the exception handlers in MIPS.
3. Try to use exceptions to make the program strong.

二、 实验内容

Implement an arithmetic calculator which can conduct addition and multiplication on two integers, which is input by the user.

1. In the following situation an exception will be triggered:
    1. the addition overflow.
    2. the multiplication result exceeds the width of a word.
2. The exception handler('trap' is sugguested) should do the following things:
    1. stop the program running.
    2. output prompt information, including "runtime exception at 0x_(the address of the instruction which triggered the exception)", and the cause of the exception (the sum is overflow, the product is bigger than the Max value of a word).
    3. exit the program.

三、 实验步骤（阐述代码思路或操作步骤）

Problem 1

> If the operator user input is not `*` or `+`, then print out the prompt and exit the program. If the integer user input is not in range of 31-bit binary number, then the exception trapped and prompt user the related information with re-input. If the sumation is more than 31-bit, then exception trapped with related information and program exit. If the product is more than 32-bit, then exception trapped with related information and program exit.

```
.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

# operand_1     $s0
# operator      $s1
# operand_2     $s2
# Exception     $s4    1: exception

    .text

    .global main
    main:
    print_string("Welcome to use the simple arithmetic calculator on unsigned 31-bit
number:\n")
    print_string("Please input operator:> ")
    li      $v0, 12  # read character $v0 contains character read   +:43 *42
    syscall
    add     $s1, $v0, $zero     # $s1 = $v0 + $zero
    li      $t0, 42
    beq     $s1, $t0, multiply  # if $s1 == $t1 then target
    li      $t0, 43
    beq     $s1, $t0, addition  # if $s1 == $t1 then target
    print_string("\nThe operator ")
    move    $a0, $s1
    li      $v0, 11             # system call to print character in $a0
    syscall
    print_string(" is not supported ,exit")
    end

    multiply:
        print_string("\nPlease input multiplicand:> ")
        jal     get             # jump to get and save position to $ra
        move    $s0, $v0
        print_string("Please input multiplier:> ")
        jal     get             # jump to get and save position to $ra
        move    $s2, $v0
        mult    $s0, $s2            # $s0 * $s2 = Hi and Lo registers
        mfhi    $t0                 # copy Lo to $t2
        tne     $t0, $zero

        print_string("The product of ")
        move    $a0, $s0
        li      $v0, 1                  # print integer, $a0 = integer to print
        syscall
        print_string(" and ")
        move    $a0, $s2
        li      $v0, 1                  # print integer, $a0 = integer to print
        syscall
        print_string(" is: ")
        mflo    $a0
        li      $v0, 36                  # print integer as unsigned, $a0 = integer to
print
```

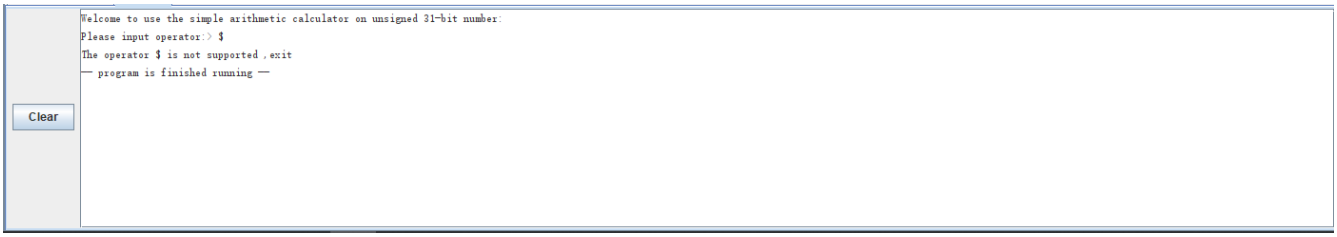
```
        syscall
        end
    addition:
        print_string("\nPlease input addend:> ")
        jal     get             # jump to get and save position to $ra
        move    $s0, $v0
        print_string("Please input augend:> ")
        jal     get             # jump to get and save position to $ra
        move    $s2, $v0

        add    $t0, $s0, $s2
        print_string("The sum of ")
        move    $a0, $s0
        li      $v0, 1                  # print integer, $a0 = integer to print
        syscall
        print_string(" and ")
        move    $a0, $s2
        li      $v0, 1                  # print integer, $a0 = integer to print
        syscall
        print_string(" is: ")
        move    $a0, $t0
        li      $v0, 1                  # print integer, $a0 = integer to print
        syscall
        end
    get:
        move    $t0, $ra
        li      $v0, 5                  # read integer $v0 contains integer read
        syscall
        tlt     $v0,  $zero              # input_negetive_error

        move    $ra, $t0

        jr      $ra                 # jump to $ra

    .ktext 0x80000180
        mfc0    $k0,$14        # $14 (epc) address of instruction that caused exception
        la      $k1, get
        addi    $k1, $k1, 8     # address of syscall in get
        beq     $k1, $k0, input_overflow    # if the error is related to the read-integer,
then output the input error
        addi    $k1, $k1, 4
        beq     $k1, $k0, input_negetive
        la      $k1, addition
        addi    $k1, $k1, 48
        beq     $k1, $k0, addition_overflow
        la      $k1, multiply
        addi    $k1, $k1, 56
        beq     $k1, $k0, product_overflow
        product_overflow:
        la      $a0, error_1
        li      $v0, 4
        syscall
        mfc0    $a0, $14
```

```
        li      $v0, 34            # Displayed value is 8 hexadecimal digits, left-padding
with zeroes if necessary.
        syscall
        la      $a0, product_overflow_error
        li      $v0, 4
        syscall
        end


        addition_overflow:
        la      $a0, error_1
        li      $v0, 4
        syscall
        mfc0    $a0, $14
        li      $v0, 34            # Displayed value is 8 hexadecimal digits, left-padding
with zeroes if necessary.
        syscall
        la      $a0, addition_overflow_error
        li      $v0, 4
        syscall
        end
        # break       1            # terminate program execution with exception
        input_negetive:
        la      $a0, input_negetive_error
        li      $v0, 4
        syscall
        la      $ra, get
        addi    $ra, $ra, 4             # prevent restore $ra in method named get (which
will cause bug)
        jr      $ra                     # jump to $ra
        input_overflow:
        la      $a0, input_overflow_error
        li      $v0, 4
        syscall
        la      $ra, get
        addi    $ra, $ra, 4             # prevent restore $ra in method named get (which
will cause bug)
        jr      $ra                     # jump to $ra

    .kdata
        msg: .asciiz "\nInput overflow"
        input_overflow_error: .asciiz "\n> Please input an integer which is between 0 and
0x7FFFFFFF.\n> "
        input_negetive_error: .asciiz "\n> Please input an integer which is bigger than
0.\n> "
        error_1: .ascii "\nRuntime exception at \0"
        addition_overflow_error: .asciiz ", the sum is overflow."
        product_overflow_error: .asciiz ", the product is bigger than the Max value of a
word."
```

四、 实验结果（截图并配以适当的文字说明）

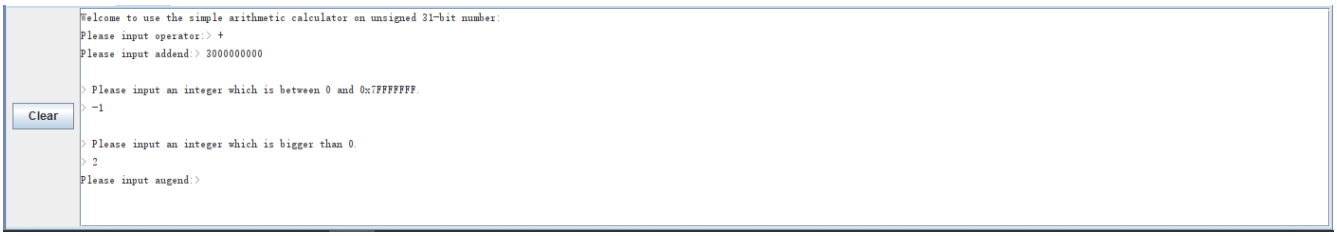
Problem 1

1. Operator Error



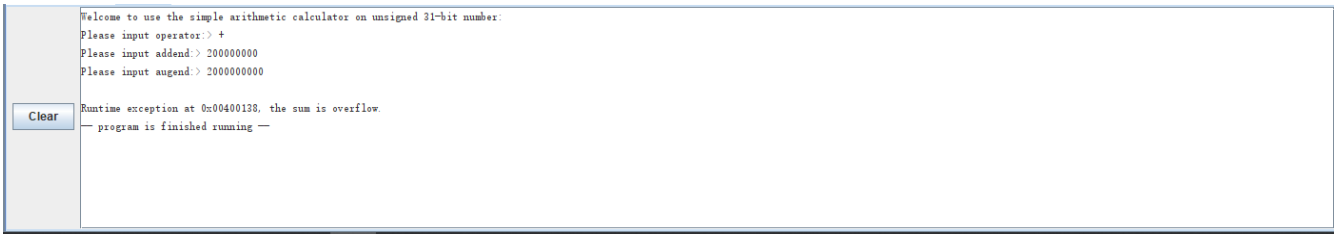
If the operator is not * or +, then output the prompt and exit the program.

2. Input not in range



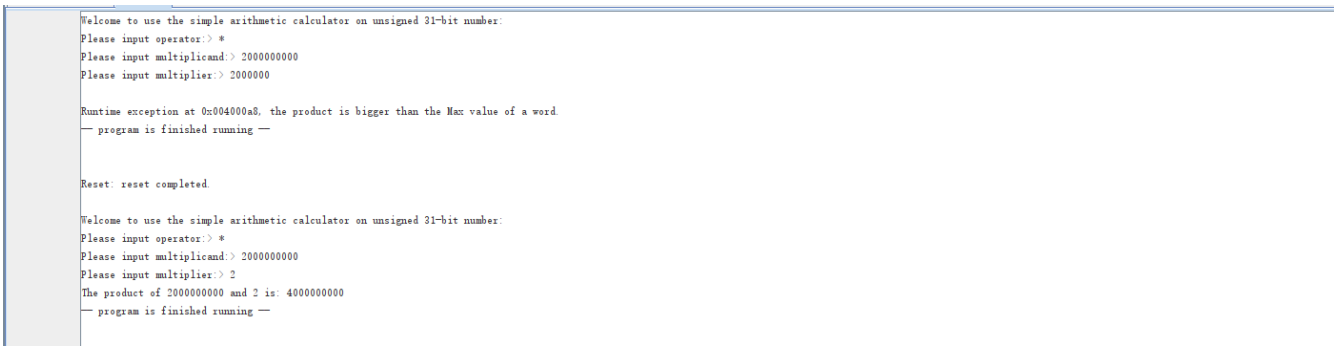
If the input is not unsigned 31-bit number, then the exception trapped and prompt user the related information with re-input.

3. Summation overflow



If the sumation is more than 31-bit, then exception trapped with related information and program exit.

4. Product exceeds the width of a word.



If the product is more than 32-bit, then exception trapped with related information and program exit.

五、 实验分析（遇到的问题以及解决方案）

1. To differ the different exception is not a easy work.

- Use relative addresses to distinguish between different exceptions

六、 实验小结与体会

1. A good programmer should make good use of exception handler.