BigInteger Library in MIPS-32 Assembly

November 2018

CS321/CS322 Mini-Project Report

Mukuntha N S
1601CS27

Introduction

The primitive integer data type is commonly limited to a size of 32-bits or 64-bits in most programming languages (eg. C, C++, Java). This means that calculations involving very large integers cannot be directly handled with these primitive data types. So for example, a number such as the factorial of 100, which contains 158 characters, can't be handled directly. So, several programming languages have big-integer libraries to handle such big integer operations (Java has the BigInteger Library, C++ has boost libraries to support big integers, python automatically handles these as 'bignum' integers). In this project, I have implemented several basic functions in the MIPS-32 instruction set, to be run on the MARS (Mips Assembly and Runtime Simulator) emulator. **To the best of my knowledge, there is currently no implementation of a big integer library available in the MIPS-32 instruction set, and this is the first. The few implementations found online were for C/C++ and were part of much bigger libraries. My implementation also uses word-by-word operations and should thus be much more efficient.**

Description

In this project, I have implemented the following functions for handling big integers. Similar to the way C handles structs, a big integer is stored with the first 32-bits with an unsigned integer storing the size of the big integer. The rest is an integer array storing the big integer. The least-significant-bit is stored first (little-endian).

This code currently supports a big integer of a maximum size of 2^32 bytes. All operations work on arbitrary length big integers. The code currently only supports unsigned operations. It can easily be extended to include signed operations, by handling signs externally and using unsigned operations in the right order.

Below, I've provided a brief explanation (documentation) of all the functions implemented:

`make_bi`

This function makes a big integer of a given length in bytes (passed through $a0). The size is rounded-off to the nearest 4 multiple and a new big integer is allotted. The first 4 bytes store the size and the rest is an integer array of the same size in bytes. The starting address of this big integer is returned in $v0. The allotting is done dynamically using the MARS syscall for dynamic allocation.

`make_bi_100`

This function calls the make_bi function and returns a big integer with a size of 100 bytes.

`ascii_to_int`

This function converts a hexadecimal ASCII character passed to it in $a0 to an integer and returns it in $v0.

`make_bi_from_str`

This function takes in the address of a string representing a hexadecimal number and its length and converts it to a big integer.

`add_3_words`

This function is used to add three 32-bit integers and handle the carry. $a0, $a1, $a2 are added. Return: $v0 - Lower half of sum, $a0 - Upper half of sum (carry)

`print_bi`

This function prints a big integer, using the MARS syscall for printing 32-bit hexadecimal integers. $a0 - address of bi to print.

`set_bi_zero`

$a0 - address of bi to set to zero. The size is read, and the entire big integer is set to 0.

`add_bi_bi`

This is a function that adds 2 big integers. Unsigned addition is done word-by-word. Returns $v0 - address of new big integer. The function first makes a new big integer to store the result of size max(l1, l2) + 4 bytes where l1, l2 are the sizes in bytes of the input big integers. The addition is done word-by-word, and the carry is stored. The function calls the add_3_words subroutine for adding the two words and the carry word.

`sub_bi_bi`

This is a function that finds the difference of 2 big-integers. Unsigned subtraction is done byte-by-byte. The smaller big-integer is subtracted from the smaller big-integer. Returns $v0 - address of new big-integer. The comparison function is used to determine if the big integers need to be swapped. The function then makes a new big integer to store the result of size max(l1, l2) bytes where l1, l2 are the sizes in bytes of the input big integers. The subtraction is done byte-by-byte, and the borrow is stored if needed. When doing the subtraction, the borrow is added to the subtrahend, and then this sum is subtracted from the minuend (minuend - (subtrahend+borrow) = difference). This is done byte-by-byte since the sum of the subtrahend and the borrow might not fit in a 32-bit register if done word-by-word.

`mult_bi_bi`

This is a function that adds 2 big integers. Unsigned multiplication is done word-by-word. Returns $v0 - address of new big integer.

The outer loop loops through the first big integer. The inner loop loops through the second. A new big integer of size l1 + l2 is made and initialized to 0. Each word from the first big integer gets multiplied to the whole second big integer, with the carry handled. Again, the add_3_words subroutine is used to add the carry, the accumulated sum in the result, and the product of the words. This is then put in the right position in the result.

`comp_bi_bi`

This is a function that compares two big integers. It returns boolean flags for ‘equal’, ‘less-than’, ‘greater-than’, ‘less-than-or-equal’, and ‘greater-than-or-equal’. Values passed - $a0 - first bi, $a1 - second bi. Returns: $v0 - Equal, $a0 - LT, $a1 - GT, $a2 - LTE, $a3 - GTE. First, the lengths of the two big integers are compared. If they’re equal, the integers are compared word-by-word, and the flags are set depending on the comparison.

Specifications

The following code is meant to run on the MIPS-32 bit. The code and some sample output are provided in the following pages.

Code

```
.data

bi_a: .asciiz "1111111111111"
bi_b: .asciiz "22222222222222222"

a_is: .asciiz "A is:\n"
b_is: .asciiz "B is:\n"

equal_str: .asciiz "A == B is "
lt_str: .asciiz "A < B is "
lte_str: .asciiz "A <= B is "
gt_str: .asciiz "A > B is "
gte_str: .asciiz "A >= B is "

true_str: .asciiz "True\n"
false_str: .asciiz "False\n"

sum_str: .asciiz "Sum is: \n"
diff_str: .asciiz "Difference is: \n"
prod_str: .asciiz "Product is: \n"


.text
j main

# Initializes a new big integer
# $a0 - no. of bytes
```

```
make_bi:
     # addi $a0, $a0, 1
     addi $a0, $a0, -1
     div $a0, $a0, 4
     mulu $a0, $a0, 4
     # convert to next 4 multiple, add 4 for storing size
     addi $a0, $a0, 8
     li   $v0,9                # To allocate a block of memory
     syscall                   # $v0 <-- address
     addi $a0, $a0, -4
     sw $a0, 0($v0)
     jr $ra


# Initializes a new big integer 100 bytes long
make_bi_100:
     li $a0, 100
     addi $sp, $sp, -4
     sw $ra, 0($sp)
     jal make_bi
     lw $ra, 0($sp)
     addi $sp, $sp, 4
     jr $ra


# $a0 - stores character to convert
ascii_to_int:
     li $t1, 97
     sltu $t0, $a0, $t1
     bne $t0, 0, ascii_to_int_if2
     addi $v0, $a0, -87
```

```
    jr $ra

    ascii_to_int_if2:
    li $t1, 65
    sltu $t0, $a0, $t1
    bne $t0, 0, ascii_to_int_if3
    addi $v0, $a0, -55
    jr $ra

    ascii_to_int_if3:
    addi $v0, $a0, -48
    jr $ra


# Makes a big integer from a string
# $a0 - address of string
# $a1 - length of string
make_bi_from_str:
    # $t0 - bi
    addi $sp, $sp, -32
    sw $ra, 0($sp)
    sw $s0, 4($sp)
    sw $s1, 8($sp)
    sw $s2, 12($sp)
    sw $s3, 16($sp)
    sw $s4, 20($sp)
    sw $s5, 24($sp)
    sw $s6, 28($sp)
    move $s0, $a0
    move $s1, $a1
```

```
# make_bi with half the length given
addi $a0, $a1, 1
div $s4, $a0, 2
move $a0, $s4
jal make_bi
# Save address of bi in $s3
move $s3, $v0
# get size in $s2
lw $s2, 0($s3)

# s4 - points to the last byte in array
add $s4, $v0, $s4
addi $s4, $s4, 3


# Parity bit for loop
andi $s5, $s1, 1
# xori $s5, 1
# move $s5, $0

# Index of character to copy
move $s6, $0

make_bi_from_str_loop_begin:
     # if $s1 <= 0, break_loop
     slti $t0, $s1, 1
     # if ($s1 < 1) != 0, break_loop
     bne $t0, $0, make_bi_from_str_break_loop
     addi $s1, $s1, -1
     # t2 - Address of character to copy
     add $t2, $s0, $s6
```

```
        addi $s6, $s6, 1
        # Load char into $a0
        lb $a0, 0($t2)
        # Convert to int
        jal ascii_to_int
        # Store int in the last byte of array if parity bit is
0
        bne $s5, $0, make_bi_from_str_else1
            lb $t0, 0($s4)
            mulu $v0, $v0, 16
            add $t0, $t0, $v0
            sb $t0, 0($s4)
            li $s5, 1
            j make_bi_from_str_after_else1
        # Else store it in the upper half of the byte
        make_bi_from_str_else1:
            lb $t0, 0($s4)
            add $t0, $t0, $v0
            sb $t0, 0($s4)
            addi $s4, $s4, -1
            li $s5, 0
        make_bi_from_str_after_else1:

        j make_bi_from_str_loop_begin

    make_bi_from_str_break_loop:
    # move $t0, $v0
    # Return address of bi
    move $v0, $s3
    lw $s6, 28($sp)
    lw $s5, 24($sp)
```

```
     lw $s4, 20($sp)
     lw $s3, 16($sp)
     lw $s2, 12($sp)
     lw $s1, 8($sp)
     lw $s0, 4($sp)
     lw $ra, 0($sp)
     addi $sp, $sp, 32
     jr $ra

# $a0, $a1, $a2 are added
# $v0 - Lower half of sum
# $a0 - Upper half of sum (carry)
add_3_words:
     addu $t1, $a0, $a1
     sltu $t0, $t1, $a0  # set carry-in bit
     addu $v0, $t1, $a2
     sltu $t2, $v0, $t1  # set carry-in bit
     addu $a0, $t0, $t2   # Add carry bits
     jr $ra


# $a0 - address of bi
print_bi:
     addi $sp, $sp, -12
     sw $ra, 0($sp)
     sw $s0, 4($sp)
     sw $s1, 8($sp)
     move $s1, $a0
     # load last word address into $s0
     lw $s0, 0($s1)
     add $s0, $s0, $s1
```

```
     # while s1 < s0, print 0($s0)
     print_bi_loop1:
     sltu $t0, $s1, $s0
     beq $t0, $0, print_bi_break_loop
     # Print last word
     lw $a0, 0($s0)
     li $v0, 34
     syscall
     # Print a space between words
     li $a0, ' '
     li $v0, 11
     syscall
     addi $s0, $s0, -4
     j print_bi_loop1
     print_bi_break_loop:
     lw $s1, 8($sp)
     lw $s0, 4($sp)
     lw $ra, 0($sp)
     addi $sp, $sp, 12
     jr $ra


# $a0 - address of bi
set_bi_zero:
     addi $sp, $sp, -12
     sw $ra, 0($sp)
     sw $s0, 4($sp)
     sw $s1, 8($sp)
     move $s1, $a0
     # load last byte address into $s0
     lw $s0, 0($s1)
```

```
    add $s0, $s0, $s1
    # while s1 < s0, set 0($s0) to 0
    set_bi_loop1:
    sltu $t0, $s1, $s0
    beq $t0, $0, set_bi_break_loop
    # Set last byte to 0
    sw $0, 0($s0)
    addi $s0, $s0, -4
    j set_bi_loop1
    set_bi_break_loop:
    lw $s1, 8($sp)
    lw $s0, 4($sp)
    lw $ra, 0($sp)
    addi $sp, $sp, 12
    jr $ra


# $a0 - first bi
# $a1 - second bi
add_bi_bi:
    addi $sp, $sp, -32
    sw $ra, 0($sp)
    sw $s0, 4($sp)
    sw $s1, 8($sp)
    sw $s2, 12($sp)
    sw $s3, 16($sp)
    sw $s4, 20($sp)
    sw $s5, 24($sp)
    sw $s6, 28($sp)

    move $s5, $a0
```

```
move $s6, $a1

# find sizes of bi
lw $s0, 0($a0)
lw $s1, 0($a1)
# put the bigger of the sizes in $s3
move $s3, $s1
sltu $t0, $s3, $s0
# if $s3 not less than $s0, goto skip_if1
beq $t0, $0, add_bi_bi_skip_if1
move $s3, $s0
add_bi_bi_skip_if1:
addi $s3, $s3, 4
# make a new bi with this size + 4
move $a0, $s3
jal make_bi
move $s4, $v0
lw $s3, 0($v0)
# Make byte-sizes word-sizes
div $s0, $s0, 4
div $s1, $s1, 4
div $s3, $s3, 4

# Loop counter
move $s2, $0
move $a0, $0
# while s2 < s3
add_bi_bi_loop_begin:
sltu $t0, $s2, $s3
beq $t0, $0, add_bi_bi_break_loop
     # $a1, $a2 <= lw((4 * $s2 + 4)+($s<>))
```

```
        # $a0 <= carry
        mulu $t0, $s2, 4
        addi $t0, $t0, 4
        move $a1, $0
        move $a2, $0
        # if s2 < s0, lw from $s5
        sltu $t1, $s2, $s0
        beq $t1, $0, add_bi_bi_skip_else1
             add $t1, $t0, $s5
             lw $a1, 0($t1)
        add_bi_bi_skip_else1:
        # if s2 < s1, lw from $s6
        sltu $t1, $s2, $s1
        beq $t1, $0, add_bi_bi_skip_else2
             add $t1, $t0, $s6
             lw $a2, 0($t1)
        add_bi_bi_skip_else2:
        # Store the sum in the new bi
        addi $sp, $sp, -4
        sw $t0, 0($sp)
        jal add_3_words
        lw $t0, 0($sp)
        addi $sp, $sp, 4
        # add $t1, $v0, $t0
        # add $t1, $t1, $s4
        # sw $t0, 0($t1)
        add $t1, $t0, $s4
        sw $v0, 0($t1)
        addi $s2, $s2, 1
        j add_bi_bi_loop_begin
   add_bi_bi_break_loop:
```

```
    move $v0, $s4
    lw $s6, 28($sp)
    lw $s5, 24($sp)
    lw $s4, 20($sp)
    lw $s3, 16($sp)
    lw $s2, 12($sp)
    lw $s1, 8($sp)
    lw $s0, 4($sp)
    lw $ra, 0($sp)
    addi $sp, $sp, 32
    jr $ra


# $a0 - first bi
# $a1 - second bi
sub_bi_bi:
    addi $sp, $sp, -36
    sw $ra, 0($sp)
    sw $s0, 4($sp)
    sw $s1, 8($sp)
    sw $s2, 12($sp)
    sw $s3, 16($sp)
    sw $s4, 20($sp)
    sw $s5, 24($sp)
    sw $s6, 28($sp)
    sw $s7, 32($sp)

    move $s0, $a0
    move $s1, $a1

    # $a0, $a1 - still have the bi's
```

```
jal comp_bi_bi
# If not (bi1 < bi2), dont swap them
beq $a0, $0, sub_bi_bi_skip_swap_bis

# swap $s0 and $s1
xor $s0, $s0, $s1
xor $s1, $s0, $s1
xor $s0, $s0, $s1

sub_bi_bi_skip_swap_bis:
# find sizes of bi
lw $s3, 0($s0) # l1
lw $s4, 0($s1) # l2

# make a new bi with size l1
move $a0, $s3
jal make_bi
move $s2, $v0

# s0, s1, s2 : bi_1, bi_2, result_bi
# s3 : l1; s4 : l2
# (s5) i = 0
move $s5, $0
# (s6) borrow = 0
move $s6, $0

# while i < l1
sub_bi_bi_loop_begin:
sltu $t0, $s5, $s3
# Break if i < l1 isn't true (if i < l1 == 0)
beq $t0, $0, sub_bi_bi_loop_break
```

```
# (t3) n1 = s0[i]
addu $t3, $s0, $s5
addi $t3, $t3, 4
lb $t3, 0($t3)

# (t4) n2 = 0
move $t4, $0
# if i < l2:
sltu $t0, $s5, $s4
beq $t0, $0, sub_bi_bi_skip_copy_s1_i
     # (t4) n2 = s1[i]
     addu $t4, $s1, $s5
     addi $t4, $t4, 4
     lb $t4, 0($t4)
sub_bi_bi_skip_copy_s1_i:

# n2 += borrow
addu $t4, $t4, $s6

# if n1 >= n2 ::: if not(n1 < n2)
sltu $t0, $t3, $t4
bne $t0, $0, sub_bi_bi_else
     # n1 - n2
     subu $t1, $t3, $t4
     # address of s2[i]
     add $t5, $s2, $s5
     addi $t5, $t5, 4
     # s2[i] = n1 - n2
     sb $t1, 0($t5)
     # borrow = 0
     move $s6, $0
```

```
    j sub_bi_bi_skip_else
sub_bi_bi_else:
# else:
    # 256 + n1 - n2
    addi $t1, $t3, 256
    subu $t1, $t1, $t4
    # address of s2[i]
    add $t5, $s2, $s5
    addi $t5, $t5, 4
    # s2[i] = 256 + n1 - n2
    sb $t1, 0($t5)
    # borrow = 1
    li $s6, 1
sub_bi_bi_skip_else:
# i++
addi $s5, $s5, 1
j sub_bi_bi_loop_begin
sub_bi_bi_loop_break:

# Return $s2
move $v0, $s2

lw $s7, 32($sp)
lw $s6, 28($sp)
lw $s5, 24($sp)
lw $s4, 20($sp)
lw $s3, 16($sp)
lw $s2, 12($sp)
lw $s1, 8($sp)
lw $s0, 4($sp)
lw $ra, 0($sp)
```

```
    addi $sp, $sp, 36
    jr $ra


# $a0 - first bi
# $a1 - second bi
mult_bi_bi:
    addi $sp, $sp, -36
    sw $ra, 0($sp)
    sw $s0, 4($sp)
    sw $s1, 8($sp)
    sw $s2, 12($sp)
    sw $s3, 16($sp)
    sw $s4, 20($sp)
    sw $s5, 24($sp)
    sw $s6, 28($sp)
    sw $s7, 32($sp)

    move $s0, $a0
    move $s1, $a1
    # get the two sizes
    lw $s3, 0($s0)
    lw $s4, 0($s1)
    # make new bi with sum of sizes
    add $a0, $s3, $s4
    jal make_bi
    move $s2, $v0
    lw $s5, 0($s2)
    move $a0, $s2
    jal set_bi_zero
    # # Convert sizes to words
```

```
# div $s3, $s3, 4
# div $s4, $s4, 4
# div $s5, $s5, 4

# $s0, $s1, $s2 - addresses of the bi's
## addi $s0, $s0, 1
## addi $s1, $s1, 1
## addi $s2, $s2, 1
# $s3, $s4, $s5 - sizes of the bi's
# $s6, $s7 - i, j - loop vars
li $s6, 0
# while $s6 < $s3, i++
mult_bi_bi_outer_loop_begin:
sltu $t0, $s6, $s3
beq $t0, $0, mult_bi_bi_after_outer_loop
     # incrementing in the beginning,
     #  since array access is 1 indexed
     addi $s6, $s6, 4
     add $t1, $s0, $s6
     # t2 - digit from s0
     lw $t2, 0($t1)
     # a0 - prev_carry
     move $a0, $0
     # j=0
     move $s7, $0
     # while $s7 < $s3, ++j
     mult_bi_bi_inner_loop_begin:
     sltu $t0, $s7, $s4
     beq $t0, $0, mult_bi_bi_after_inner_loop
          # incrementing in the beginning,
          #  since array access is 1 indexed
```

```
addi $s7, $s7, 4
add $t1, $s1, $s7
# a2 - digit from s1
lw $a2, 0($t1)
# a2 - lo of product
mulu $a2, $t2, $a2
# t5 - hi of product
mfhi $t5
# a1 : address and then data of the
#  [i + j - 1]'th element of the result bi
add $a1, $s2, $s6
add $a1, $a1, $s7
addi $a1, $a1, -4
# load data
lw $a1, 0($a1)
addi $sp, $sp, -8
sw $t2, 0($sp)
sw $t5, 4($sp)
jal add_3_words
# Now the new carry is still in $a0
lw $t5, 4($sp)
lw $t2, 0($sp)
addi $sp, $sp, 8
# Put sum in [i + j - 1]'th element of the result
bi
add $a1, $s2, $s6
add $a1, $a1, $s7
addi $a1, $a1, -4
sw $v0, 0($a1)

# Add carry from mulu to current carry
```

```
                add $a0, $a0, $t5
            j mult_bi_bi_inner_loop_begin
            mult_bi_bi_after_inner_loop:

            # Store carry in last element of result
            add $a1, $s2, $s6
            add $a1, $a1, $s7
            sw $a0, 0($a1)

            j mult_bi_bi_outer_loop_begin
        mult_bi_bi_after_outer_loop:

        # Return the address of the new bi
        move $v0, $s2

        lw $s7, 32($sp)
        lw $s6, 28($sp)
        lw $s5, 24($sp)
        lw $s4, 20($sp)
        lw $s3, 16($sp)
        lw $s2, 12($sp)
        lw $s1, 8($sp)
        lw $s0, 4($sp)
        lw $ra, 0($sp)
        addi $sp, $sp, 36
        jr $ra


# $a0 - first bi
# $a1 - second bi
# Return:
```

```
# $v0 - Equal, $a0 - LT, $a1 - GT
# $a2 - LTE, $a3 - GTE
comp_bi_bi:
     addi $sp, $sp, -24
     sw $ra, 0($sp)
     sw $s0, 4($sp)
     sw $s1, 8($sp)
     sw $s2, 12($sp)
     sw $s3, 16($sp)
     sw $s4, 20($sp)

     move $s0, $a0
     move $s1, $a1
     # get the two sizes
     lw $s2, 0($s0)
     lw $s3, 0($s1)

     # Compare lengths, return if one is greater
     sltu $t0, $s2, $s3
     bne $t0, $0, comp_bi_bi_lt_return
     sltu $t0, $s3, $s2
     bne $t0, $0, comp_bi_bi_gt_return

     # This code runs if lengths are equal
     # i = l1
     move $s4, $s2
     comp_bi_bi_loop_begin:
     # if i <= 0 (if (i > 0) == 0), return 'equal'
          sgtu $t0, $s4, $0
          beq $t0, $0, comp_bi_bi_eq_return
          # ith word from first bi
```

```
        add $t1, $s0, $s4
        lw $t1, 0($t1)
        # ith word from second bi
        add $t2, $s1, $s4
        lw $t2, 0($t2)
        # if $t1 < $t2, return lt
        sltu $t0, $t1, $t2
        bne $t0, $0, comp_bi_bi_lt_return
        # if $t2 < $t1, return gt
        sltu $t0, $t2, $t1
        bne $t0, $0, comp_bi_bi_gt_return

        # else continue looping (they're equal)
        addi $s4, $s4, -4
        j comp_bi_bi_loop_begin

    comp_bi_bi_lt_return:
    li $v0, 0 # Eq
    li $a0, 1 # LT
    li $a1, 0 # GT
    li $a2, 1 # LTE
    li $a3, 0 # GTE
    j comp_bi_bi_return

    comp_bi_bi_gt_return:
    li $v0, 0 # Eq
    li $a0, 0 # LT
    li $a1, 1 # GT
    li $a2, 0 # LTE
    li $a3, 1 # GTE
    j comp_bi_bi_return
```

```
    comp_bi_bi_eq_return:
    li $v0, 1 # Eq
    li $a0, 0 # LT
    li $a1, 0 # GT
    li $a2, 1 # LTE
    li $a3, 1 # GTE
    j comp_bi_bi_return

    comp_bi_bi_return:

    lw $s4, 20($sp)
    lw $s3, 16($sp)
    lw $s2, 12($sp)
    lw $s1, 8($sp)
    lw $s0, 4($sp)
    lw $ra, 0($sp)
    addi $sp, $sp, 24
    jr $ra


newline:
    addi $sp, $sp, -4
    sw $ra, 0($sp)
    # Print a newline
    li $a0, '\n'
    li $v0, 11
    syscall
    lw $ra, 0($sp)
    addi $sp, $sp, 4
    jr $ra
```

```
# Find the string length of a null-terminated string
# $a0 - string address
str_len:
     addi $sp, $sp, -4
     sw $ra, 0($sp)

     move $t0, $a0
     str_len_loop:
          lb $t1, 0($t0)
          beq $t1, $0, str_len_end
          addi $t0, $t0, 1
          j str_len_loop

     str_len_end:
     subu $v0, $t0, $a0

     lw $ra, 0($sp)
     addi $sp, $sp, 4
     jr $ra


# $a0 - if 0, prints false, else true
print_true_false:
     addi $sp, $sp, -4
     sw $ra, 0($sp)

     beq $a0, $0, print_false
          la $a0, true_str
     j skip_print_false
```

```
    print_false:
         la $a0, false_str
    skip_print_false:
    addi $v0, $0, 4
    syscall

    lw $ra, 0($sp)
    addi $sp, $sp, 4
    jr $ra


main:
    # Make big integers from strings bi_a, and bi_b
    la $a0, bi_a
    jal str_len
    move $a1, $v0
    la $a0, bi_a
    jal make_bi_from_str
    move $s0, $v0

    la $a0, bi_b
    jal str_len
    move $a1, $v0
    la $a0, bi_b
    jal make_bi_from_str
    move $s1, $v0

    # Print bi_a

    la $a0, a_is
    addi $v0, $0, 4
```

```
syscall

move $a0, $s0
jal print_bi
jal newline

# Print bi_b

la $a0, b_is
addi $v0, $0, 4
syscall

move $a0, $s1
jal print_bi
jal newline

# Compare a, b
move $a0, $s0
move $a1, $s1
jal comp_bi_bi
move $t0, $v0
move $t1, $a0
move $t2, $a1
move $t3, $a2
move $t4, $a3

la $a0, equal_str
addi $v0, $0, 4
syscall

move $a0, $t0
```

```
jal print_true_false

la $a0, lt_str
addi $v0, $0, 4
syscall

move $a0, $t1
jal print_true_false

la $a0, gt_str
addi $v0, $0, 4
syscall

move $a0, $t2
jal print_true_false

la $a0, lte_str
addi $v0, $0, 4
syscall

move $a0, $t3
jal print_true_false

la $a0, gte_str
addi $v0, $0, 4
syscall

move $a0, $t4
jal print_true_false

# Sum of a, b
```

```
la $a0, sum_str
addi $v0, $0, 4
syscall

move $a0, $s0
move $a1, $s1
jal add_bi_bi
move $a0, $v0
jal print_bi
jal newline

# Difference of a, b

la $a0, diff_str
addi $v0, $0, 4
syscall

move $a0, $s0
move $a1, $s1
jal sub_bi_bi
move $a0, $v0
jal print_bi
jal newline

# Product of a, b

la $a0, prod_str
addi $v0, $0, 4
syscall
```

```
    move $a0, $s0
    move $a1, $s1
    jal mult_bi_bi
    move $a0, $v0
    jal print_bi
    jal newline
```

Output:

```
A is:
0x00011111 0x11111111
B is:
0x00000002 0x22222222 0x22222222
A == B is False
A < B is True
A > B is False
A <= B is True
A >= B is False
Sum is:
0x00000000 0x00000002 0x22233333 0x33333333
Difference is:
0x00000002 0x22211111 0x11111111
Product is:
0x00000000 0x0002468a 0xcf13579b 0xbbbb9753 0x0eca8642
```