ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ «ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Московский институт электроники и математики им. А.Н. Тихонова  
Департамент компьютерной инженерии

Курс: Вычислительные системы и компьютерные сети

Отчёт   
по практической работе №5 «Специальные типы данных: векторные операции, двоично-десятичная арифметика.»

Студент: Омаров Марат Тимурович  
Группа: БИВ206  
Вариант: 8  
Дата: 20.10.2022

Москва 2022

Оглавление

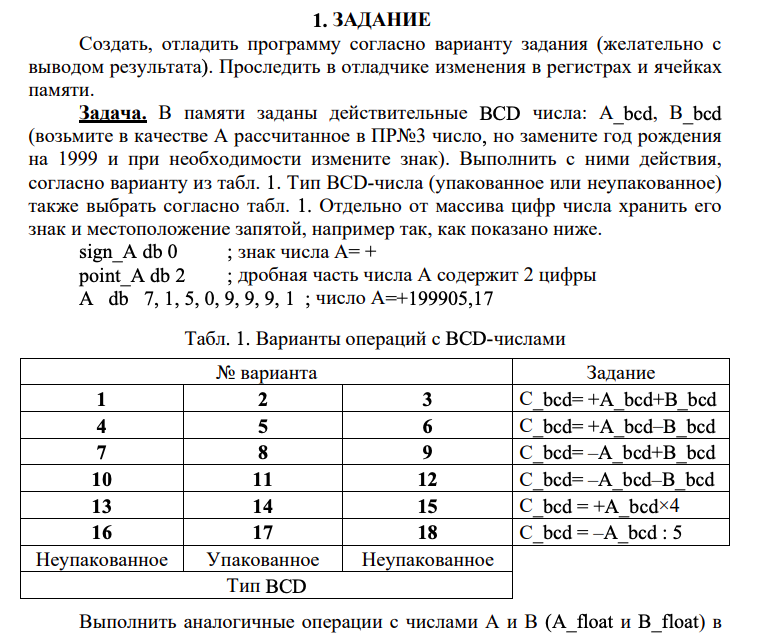
[Условие задачи 3](#_Toc117142127)

[Алгоритм выполнения 3](#_Toc117142128)

[Код MIPS 4](#_Toc117142129)

[Результат выполнения программы 11](#_Toc117142130)

# Условие задачи



# Алгоритм выполнения

Данная работа выполнялась мною в MIPS. Для реализации выполнения сложения были написаны:

* процедура по переводу упакованного типа числа в неупакованный;
* процедура вывода неупакованного числа в консоль вместе со знаком этого числа;
* процедура копирования неупакованного числа в другую переменную;
* процедура перевода неупакованного числа в обратный код для дальнейшей реализации операции вычитания;
* функция сложения двух неупакованных чисел.

На следующей странице можно наблюдать код программы.

Для удобства прилагаю ссылку на github с моим кодом: <https://github.com/marato-o/CS-CN/blob/master/lab_5/lab_5.asm>

# Код MIPS

.data

A\_bcd: .byte 0x99 0x98 0x19 # записали число 199899 в упакованном формате через шестнадцатеричную запись

A\_sign: .byte 0 # если знак отрицательный, пишем 1

A\_size: .byte 3

B\_bcd: .byte 0x05 # записали число 0503 в упакованном формате через шестнадцатеричную запись

B\_sign: .byte 0x0

B\_size: .byte 2

C\_bcd: .byte 0 0 0 0

C\_sign: .byte 0x0

C\_size: .byte 4

A\_unpack: .byte 0 0 0 0 0 0 0

A\_un\_size: .byte 7 # размер распакованного числа равен 2\*A\_size + 1, для корректного вывода сообщения с учётом знака числа

B\_unpack: .byte 0 0 0

B\_un\_size: .byte 3

C\_unpack: .byte 0 0 0 0 0 0 0 0 0

C\_un\_size: .byte 9

A\_reverse: .byte 0 0 0 0 0 0 0

B\_reverse: .byte 0 0

mes\_n: .asciiz "\n"

string: .asciiz ""

.text

# в начале реализуем процедуру перевода упакованного числа в неупакованное

Prepare:

# для передачи данных в процедуру используем регистры a

la $a0, A\_bcd

la $a1, A\_unpack

la $a2, A\_size

lb $a2, ($a2)

jal Proc\_To\_Unpack

la $a0, B\_bcd

la $a1, B\_unpack

la $a2, B\_size

lb $a2, ($a2)

jal Proc\_To\_Unpack

# начинаем подготовку к вызову процедуры сложения

# для упрощения работы сначала заполняем в переменную C большую из переменных

comparison:

la $t0, A\_unpack

la $t1, A\_un\_size

lb $t1, ($t1)

la $t2, B\_unpack

la $t3, B\_size

lb $t3, ($t3)

subu $t4, $t3, $t1

la $a1, C\_unpack

bgtz $t4, copy\_second

copy\_first:

move $v0, $t3

subu $v1, $t1, $t3

move $a0, $t0

move $a2, $t1

j continue

copy\_second:

move $v0, $t1

subu $v1, $t3, $t1

move $a0, $t2

move $a2, $t3

continue:

jal Proc\_Copy # в результате в C\_unpack запишется большая по модулю из переменных

Addition\_function:

move $t0, $v0 # записываем размерность меньшего числа

li $t1, 0 # используем как счётчик

li $t2, 0 # используем для хранения переноса

li $t3, 0 # получение байта из первой переменной

li $t4, 0 # получение байта из второй переменной

addition\_loop:

beq $t0, $t1, end\_loop

la $t3, A\_unpack # получаем адрес начала массива числа

addu $t3, $t3, $t1 # перемещаемся к адресу нужного байта

lbu $t3, ($t3) # перезаписываем в t3 значение этого байта

beq $t2, 0, without\_transfer\_from\_junior

addu $t3, $t3, $t2

without\_transfer\_from\_junior:

li $t2, 0

la $t4, B\_unpack # получаем адрес начала массива числа

addu $t4, $t4, $t1 # перемещаемся к адресу нужного байта

lbu $t4, ($t4) # перезаписываем в t3 значение этого байта

addu $t5, $t3, $t4 # записали в t5 сумму байтов двух чисел

andi $t3, $t5, 0x0F # перезаписываем в t3 младшую тетраду

subu $t4, $t3, 0xA

bltz $t4, no\_adjustment # если младшая тетрада < 10, переходим на метку

addiu $t5, $t5, 0x6 # добавляем корректирующую величину

no\_adjustment:

andi $t4, $t5, 0xF0

srl $t4, $t4, 4 # перезаписываем в t4 старшую тетраду

beq $t4, 0, no\_transfer

move $t2, $t4 # запоминаем значение переноса в регистр

andi $t5, $t5, 0x0F # обнуляем старшую тетраду

no\_transfer:

la $t6, C\_unpack

addu $t6, $t6, $t1

sb $t5, ($t6) # запоминаем в переменную C значение байта

addi $t1, $t1, 1

j addition\_loop

end\_loop:

# теперь, когда этот цикл завершился, нужно рассмотреть старшие байты большего числа на предмет переноса

beq $t2, 0, end\_function

addu $t0, $t1, $v1

last\_loop:

beq $t0, $t1, end\_function

la $t6, C\_unpack

addu $t6, $t6, $t1

lb $t7, ($t6)

beq $t2, 0, without\_transfer\_last

addu $t7, $t7, $t2

without\_transfer\_last:

li $t2, 0

andi $t3, $t7, 0x0F # перезаписываем в t3 младшую тетраду

subu $t4, $t3, 0xA

bltz $t4, no\_adjustment\_last # если младшая тетрада < 10, переходим на метку

addiu $t7, $t7, 0x6 # добавляем корректирующую величину

no\_adjustment\_last:

andi $t4, $t7, 0xF0

srl $t4, $t4, 4 # перезаписываем в t4 старшую тетраду

beq $t4, 0, no\_transfer\_last

move $t2, $t4 # запоминаем значение переноса в регистр

andi $t7, $t7, 0x0F # обнуляем старшую тетраду

no\_transfer\_last:

sb $t7, ($t6) # запоминаем в переменную C значение байта

addi $t1, $t1, 1 # увеличиваем счётчик

j last\_loop

end\_function:

la $a0, C\_unpack

la $a1, C\_un\_size

lb $a1, ($a1)

la $a2, string

la $a3, A\_sign

lb $a3, ($a3)

jal Proc\_Print

la $a0, string

la $v0, 4

syscall

Exit:

li $v0, 10

syscall

#----------- процедура копирования неупакованного числа в другую переменную --------------#

Proc\_Copy:

# принимаем три переменные: два адреса и размер

move $t0, $a0 # адрес копируемой переменной

move $t1, $a1 # адрес пустой переменной

move $t2, $a2 # размерность

li $a0, 0

li $a1, 0

li $a2, 0

li $t3, 0 # используем как счётчик

copy\_loop:

beq $t2, $t3, exit\_copy\_proc

move $t4, $t0 # получаем адрес начала массива числа

addu $t4, $t4, $t3 # перемещаемся к адресу нужного байта

lbu $t4, ($t4) # перезаписываем в t4 значение этого байта

move $t5, $t1 # получаем адрес байта, куда запишем результат

addu $t5, $t5, $t3 # перемещаемся к адресу нужного байта

sb $t4, ($t5)

addi $t3, $t3, 1

j copy\_loop

exit\_copy\_proc:

li $t0, 0

li $t1, 0

li $t2, 0

li $t3, 0

li $t4, 0

li $t5, 0

jr $ra

#----------- процедура перевода неупакованного числа в обратный код --------------#

Proc\_To\_Reverse:

move $t0, $a0 # = адресу неупакованного числа

move $t1, $a1 # = адрес обратного числа (пустой переменной)

move $t2, $a2 # - размер неупакованного числа

#subi $t2, $t2, 1 # учитываем лишний разряд для корректного вывода минуса (он нам не нужен)

li $a0, 0

li $a1, 0

li $a2, 0

li $t3, 0 # используем как счётчик

rev\_loop:

beq $t2, $t3, exit\_rev\_proc

move $t4, $t0 # получаем адрес начала массива числа

addu $t4, $t4, $t3 # перемещаемся к адресу нужного байта

lbu $t4, ($t4) # перезаписываем в t4 значение этого байта

li $t5, 9

subu $t4, $t5, $t4 # получаем обратное число

move $t5, $t1 # получаем адрес байта, куда запишем результат

addu $t5, $t5, $t3 # перемещаемся к адресу нужного байта

sb $t4, ($t5)

addi $t3, $t3, 1

j rev\_loop

exit\_rev\_proc:

li $t0, 0

li $t1, 0

li $t2, 0

li $t3, 0

li $t4, 0

li $t5, 0

jr $ra

#--------------- процедура записи неупакованного числа в строку ----------------#

Proc\_Print:

# реализуем процедуру печати числа

move $t0, $a0 # = адресу неупакованного числа

move $t1, $a1 # = размер неупакованного числа

move $t2, $a2 # = адресу переменной, в которую запишется сообщение

move $t3, $a3 # = знаку числа

li $a0, 0

li $a1, 0

li $a2, 0

li $t4, 0 # используем как счётчик

move $t7, $t1

subi $t7, $t7, 1

print\_loop:

beq $t1, $t4, end\_print\_loop

move $t5, $t0

addu $t5, $t5, $t4

lb $t5, ($t5) # получаем содержимое нужного байта – число

addu $t5, $t5, 0x30 # прибавляем к числу 30h – переводим число в символ

move $t6, $t2

addu $t6, $t6, $t7

sb $t5, ($t6)

subi $t7, $t7, 1 # уменьшаем счётчик на 1

addi $t4, $t4, 1

j print\_loop

end\_print\_loop:

# тут выводим знак числа, если он отрицательный

beq $t3, 0, exit\_print\_proc

print\_sign:

addu $t3, $t3, 0x2C # получаем знак '-'

move $t6, $t2

sb $t3, ($t6)

exit\_print\_proc:

li $t0, 0

li $t1, 0

li $t2, 0

li $t3, 0

li $t4, 0

li $t5, 0

jr $ra

#------------- процедура перевода упакованного числа в неупакованное -------------#

Proc\_To\_Unpack:

move $t0, $a0 # = адресу упакованного числа

move $t1, $a1 # = адрес неупакованного числа (пустой переменной)

move $t2, $a2 # - размер упакованного числа

li $a0, 0

li $a1, 0

li $a2, 0

li $t3, 0 # используем как счётчик

li $t4, 0 # используем как счётчик для неупакованной переменной

unpck\_loop:

beq $t2, $t3, end\_unpck\_loop

li $t5, 0 # используем для хранения байта

li $t6, 0 # t6 используем для хранения тетрады

li $t7, 0 # t7 используем для хранения актуального адреса байта unpack

move $t5, $t0 # получаем адрес начала массива числа A

addu $t5, $t5, $t3 # перемещаемся к адресу нужного байта

lbu $t5, ($t5) # перезаписываем в t5 значение этого байта

move $t7, $t1 # определяем адрес неупакованного числа, куда запишем результат

addu $t7, $t7, $t4

addi $t4, $t4, 1

andi $t6, $t5, 0x0F # выделяем младшую тетраду в регистр t6

sb $t6, ($t7) # сохраняем младшую тетраду в байт неупакованного числа

addi $t7, $t7, 1 # перемещаемся к следующему байту переменной unpack

andi $t6, $t5, 0xF0 # теперь записываем в t6 старшую тетраду

srl $t6, $t6, 4 # смещаем на 4 бита

sb $t6, ($t7) # сохраняем тетраду в байт неупакованного числа

addi $t4, $t4, 1 # снова увеличиваем счётчик для неупакованного числа

addi $t3, $t3, 1 # увеличиваем счётчик на 1

j unpck\_loop

end\_unpck\_loop:

# неупакованное число получили

# теперь выходим из проедуры, предварительно очистив все регистры

li $t0, 0

li $t1, 0

li $t2, 0

li $t3, 0

li $t4, 0

li $t5, 0

li $t6, 0

li $t7, 0

exit\_unpck\_proc:

jr $ra # выходим из процедуры

## Результат выполнения программы

После выполнения программного кода выведется следующее сообщение (см. рис. 2). На вход программы были поданы два числа:

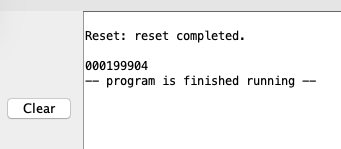


Рисунок 2. Результат выполнения программы MIPS.

Функционал по сложению двух положительных чисел работает корректно.

Для реализации вычитания достаточно в начале вызвать написанную процедуру Proc\_To\_Reverse для числа, которое необходимо сделать отрицательным, после чего уже произвести операцию сложения полученного числа с заданным. Для чисел в обратном коде операция сложения работает ровно так же, как и для положительных чисел в прямом коде. При необходимости вновь вызывается процедура Proc\_To\_Reverse для результата сложения, которая преобразует обратный код отрицательного числа в прямой.

К сожалению, полный функционал программы не реализован, в функции сложения (в основном теле программы) никак не учитывается знак числа, из-за чего проводить вычитание корректно не получится.

Для того, чтобы программа работала корректно, требуется дописать основной цикл с проверкой на перенос единицы в знаковый разряд числа, после чего, при необходимости, вызывать процедуру Proc\_To\_Reverse.

На данный момент, программа корректно выполняет сложение положительных чисел, при этом каждая из процедур работает корректно. Функционал по работе с вещественными числами не реализован.