АВС. ИДЗ-1

Гареев Данир БПИ-236

Задание

Разработать программу, которая вводит одномерный массив A, состоящий из N элементов (значение N вводится при выполнении программы), после чего формирует из элементов массива A новый массив B по правилам, указанным в варианте, и выводит его.

Вариант 40

Сформировать массив B из сумм соседних элементов A по следующему правилу:

$$B_0 = A_0, \quad B_1 = A_0 + A_1, \quad ..., \quad B_m = A_0 + ... + A_m$$

где m – номер первого элемента массива A, большего среднего арифметического этого массива. При переполнении записывать нули.

Решение на 10 баллов

Файлы хранятся на GitHub по ссылке.

Файлы:

1. **main.asm** – файл для взаимодействия с пользователем.
2. **macros.asm** – библиотека с макросами:
   - **get_size(%x)** – получает размер массива от пользователя.
   - **check_size(%x)** – проверяет корректность размера массива.
   - **add_element(%x)** – получает элемент массива от пользователя.
   - **print_element(%x)** – выводит элемент массива на экран.
   - **print_array(%array, %size)** – выводит весь массив на экран.
3. **solution.asm** – создание массива B по условиям задачи.
4. **tests.asm** – автотесты:
   - Все числа положительные
   - Все числа отрицательные
   - Все числа нули
   - Есть все 3 вида чисел
   - Есть только положительные и отрицательные
   - Есть только положительные и нули
   - Есть только отрицательные и нули
5. **user.asm** – создание массива A по условиям задачи.

Выполненные условия

На 4-5

1. Есть решение на ассембелере.
2. Есть комментарии к коду.
3. Используются подпрограммы без параметров и локальных переменных.
4. Есть отчёт с полным тестовым покрытием.

На 6-7

1. Использование подпрограмм с передачей аргументов через регистры по конвенции.
2. Сохранение локальных переменных в свободных регистрах.
3. Есть комментарии к функциям

На 8

1. Многократное использование подпрограмм.
2. Реализована доп. тестовая программа.

На 9-10

1. Используются макросы.
2. Программа разбита на несколько файлов.
3. Макросы выделены в автономную библиотеку.

Результаты автотестов

```
Enter your choice:
Enter '0': input your array.
Enter other number: launch autotests.

Your choice: 1
2 3 7
2 5 12
-2 -3 -1
-2
0 0 0
0
-4 5 0 -5 52
-4 1 1 -4 48
0 1 234 0
0 1 235
0 -1 -234 0
0
-123 1 234 -5
-123 -122 112
```

Пример работы программы

```
Enter your choice:
Enter '0': input your array.
Enter other number: launch autotests.

Your choice: 0
Input the size of the array from 1 to 10: 10
Input element: -1
Input element: 10
Input element: -62
Input element: 2
Input element: 4
Input element: 5
Input element: 1
Input element: 214
Input element: 2
Input element: 1
-1 9 -53 -51 -47 -42 -41 173
If you want to finish program, enter '0' or press any other number to restart: 1
Input the size of the array from 1 to 10: 2
Input element: 1
Input element: 2
1 3
If you want to finish program, enter '0' or press any other number to restart: 0

-- program is finished running (0) --
```

Исходный код

main.asm

```
.data
n:          .word 0
array_A:    .space 40
array_B:    .space 40

prompt_start: .asciz "\nEnter your choice:\nEnter '0': input your array.\nEnter
   other number: launch autotests.\n\nYour choice: "
prompt_next: .asciz "If you want to finish program, enter '0' or press any other
   number to restart: "
sep:        .asciz " "
newline:    .asciz "\n"

.include "macros.asm"
.include "solution.asm"
.include "user.asm"
.include "tests.asm"

.text
.global main
main:
    li   a7, 4
    la   a0, prompt_start
    ecall
    li   a7, 5
    ecall
    beqz a0, your_array
    j    to_tests
your_array:
    call user_array
    call work
    li   a7, 4
    la   a0, prompt_next
    ecall
    li   a7, 5
    ecall
    beqz a0, end
    j    your_array
```

```
to_tests:
    call autotests
end:
    li   a7, 10
    ecall
```

macros.asm

```
.macro get_size(%x)
    # Message for input array size
    li   a7, 4
    la   a0, prompt_size
    ecall
    # Read number
    li   a7, 5
    ecall
    # Set number to register %x
    mv   %x, a0
.end_macro

.macro check_size(%x)
    # Boundary values of array size
    li   t0, 1
    li   t1, 10
    # Checking
    blt  a0, t0, error
    bgt  a0, t1, error
    # Set 1 (true), if size is correct
    li   %x, 1
    j    end_check
error:
    # Message that the array size is incorrect
    li   a7, 4
    la   a0, prompt_er_size
    ecall
    # Set 0 (false), if size is incorrect
    li   %x, 0
end_check:
.end_macro

.macro add_element(%x)
    # Message for enter of element
    li   a7, 4
    la   a0, prompt_element
    ecall
    # Read element
    li   a7, 5
    ecall
    # Set element to register %x
    mv   %x, a0
.end_macro

.macro print_element(%x)
    # Print element
    li   a7, 1
    lw   a0, (%x)
    ecall
    # Print space after element
    li   a7, 4
    la   a0, sep
    ecall
.end_macro

.macro print_array(%array, %size)
    # Set a beginning of array and array size to register t0 and t3
    la   t0, %array
```

```
    li   t2, 0    # Counter of elements
    lw   t3, %size
print_el:
    beq  t2, t3, end
    # Print element
    print_element(t0)
    # Go to the next element
    addi t0, t0, 4
    addi t2, t2, 1
    j    print_el
end:
    # Print new line after array
    li   a7, 4
    la   a0, newline
    ecall
.end_macro
```

solution.asm

```
.data
b_size: .word 0

.text
work:
    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)

    # Compute 64-bit sum of A
    la   t0, array_A
    lw   s0, n           # s0 = N
    li   s1, 0           # sum_lo
    li   s2, 0           # sum_hi
    li   t1, 0           # i
sum_loop:
    beq  t1, s0, end_sum
    lw   a0, 0(t0)
    add  s1, s1, a0      # sum_lo += A[i]
    sltu t2, s1, a0      # carry out
    srai a1, a0, 31      # sign extend
    add  s2, s2, a1      # sum_hi += sign
    add  s2, s2, t2      # sum_hi += carry
    addi t0, t0, 4
    addi t1, t1, 1
    j    sum_loop
end_sum:

    # Find m
    la   t0, array_A
    lw   s0, n           # s0 = N
    li   t1, 0           # i = 0
    mv   s3, s0          # m = N initially
loop_find_m:
    bge  t1, s0, end_find_m
    lw   a0, 0(t0)
    mulh a1, a0, s0      # product_hi
    mul  a2, a0, s0      # product_lo
    bgt  a1, s2, found
    blt  a1, s2, next_i
    bgeu a2, s1, found
next_i:
    addi t0, t0, 4
```

```
    addi t1, t1, 1
    j    loop_find_m
found:
    mv   s3, t1
end_find_m:

    # Cap m to N-1 if >= N
    lw   t0, n
    bge  s3, t0, cap_m
    j    no_cap
cap_m:
    addi s3, t0, -1
no_cap:

    # Create B
    la   t0, array_A
    la   t1, array_B
    li   t2, 0           # i = 0
    li   s4, 0           # sum
    mv   s5, s3          # m_val
create_B_loop:
    bgt  t2, s5, end_create_B
    lw   a0, 0(t0)       # A[i]
    add  a1, s4, a0      # new_sum
    # Check overflow
    srai t3, s4, 31      # sum_sign
    srai t4, a0, 31      # a0_sign
    srai t5, a1, 31      # new_sign
    xor  t6, t3, t4
    bnez t6, no_overflow
    xor  t6, t3, t5
    beqz t6, no_overflow
    li   a1, 0           # overflow
    sw   zero, 0(t1)
    j    update_sum
no_overflow:
    sw   a1, 0(t1)
update_sum:
    mv   s4, a1
    addi t0, t0, 4
    addi t1, t1, 4
    addi t2, t2, 1
    j    create_B_loop
end_create_B:

    # Print B
    addi t0, s5, 1
    la   t1, b_size
    sw   t0, 0(t1)
    print_array(array_B, b_size)

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

tests.asm

```
.text
autotests:
    addi sp, sp, -4
    sw   ra, (sp)
```

```
    call store_test_1
    print_array(array_A, n)
    call work

    call store_test_2
    print_array(array_A, n)
    call work

    call store_test_3
    print_array(array_A, n)
    call work

    call store_test_4
    print_array(array_A, n)
    call work

    call store_test_5
    print_array(array_A, n)
    call work

    call store_test_6
    print_array(array_A, n)
    call work

    call store_test_7
    print_array(array_A, n)
    call work

    lw   ra, (sp)
    addi sp, sp, 4
    ret

#test 1 all positive
store_test_1:
    la   t0, array_A
    li   t1, 2
    sw   t1, 0(t0)
    li   t1, 3
    sw   t1, 4(t0)
    li   t1, 7
    sw   t1, 8(t0)
    la   t0, n
    li   t1, 3
    sw   t1, (t0)
    ret
#test 2 all neg
store_test_2:
    la   t0, array_A
    li   t1, -2
    sw   t1, 0(t0)
    li   t1, -3
    sw   t1, 4(t0)
    li   t1, -1
    sw   t1, 8(t0)
    la   t0, n
    li   t1, 3
    sw   t1, (t0)
    ret
#test 3 all zero
store_test_3:
    la   t0, array_A
    sw   zero, 0(t0)
    sw   zero, 4(t0)
    sw   zero, 8(t0)
```

```
    la   t0, n
    li   t1, 3
    sw   t1, (t0)
    ret
#test 4 zero,neg,pos
store_test_4:
    la   t0, array_A
    li   t1, -4
    sw   t1, 0(t0)
    li   t1, 5
    sw   t1, 4(t0)
    sw   zero, 8(t0)
    li   t1, -5
    sw   t1, 12(t0)
    li   t1, 52
    sw   t1, 16(t0)
    la   t0, n
    li   t1, 5
    sw   t1, (t0)
    ret
#test 5 pos,neg
store_test_5:
    la   t0, array_A
    sw   zero, 0(t0)
    li   t1, 1
    sw   t1, 4(t0)
    li   t1, 234
    sw   t1, 8(t0)
    sw   zero, 12(t0)
    la   t0, n
    li   t1, 4
    sw   t1, (t0)
    ret
#test 6 zero, pos
store_test_6:
    la   t0, array_A
    sw   zero, 0(t0)
    li   t1, -1
    sw   t1, 4(t0)
    li   t1, -234
    sw   t1, 8(t0)
    sw   zero, 12(t0)
    la   t0, n
    li   t1, 4
    sw   t1, (t0)
    ret
#test 7 zero,neg
store_test_7:
    la   t0, array_A
    li   t1, -123
    sw   t1, 0(t0)
    li   t1, 1
    sw   t1, 4(t0)
    li   t1, 234
    sw   t1, 8(t0)
    li   t1, -5
    sw   t1, 12(t0)
    la   t0, n
    li   t1, 4
    sw   t1, (t0)
    ret
```