ACS. Individual Homework 1

Александр Васюков | БПИ235 (239)

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.
Память под массивы может выделяться статически, на стеке, автоматически по выбору разработчика с учетом требований к оценке работы. При решении задачи необходимо использовать подпрограммы для реализации ввода, вывода и формирования нового массива массива. Допустимы (при необходимости) дополнительные подпрограммы. Максимальное количество элементов в массиве не должно превышать 10 (ограничение обуславливается вводом данных с клавиатуры). При этом необходимо обрабатывать некорректные значения как для нижней, так и для верхней границ массивов в зависимости от условия задачи.

Вариант 35

Сформировать массив B из элементов массива A сгруппировав положительные элементы массива в начале массива , нулевые в середине, а отрицательные – в конце

Решение, претендующее на 10 баллов:

3 раза проходимся по массиву A и добавляем сначала положительные элементы, потом нули, затем отрицательные - таким образом, сформировав массив B.

Файлы хранятся на GitHub по ссылке: https://github.com/vasyukov1/HSE-FCS-SE-2-year/tree/main/ACS/Homework/IHW_1
Есть 5 файлов:

1. main.asm - взаимодействие с пользователем. Предлагается выбор автотестов или создания собственного массива A. Также предусмотрен повторный ввод данных после выполнения программы по созданию массива B.
2. solution.asm - создание массива B. Сначала добавляются положительные элементы, потом нули, затем отрицательные элементы. После создания массив выводится на экран, после чего добавляется пустая строка для красоты.
3. user.asm - создание массива A. Сначала вводится размер массива, после чего он проверяется на корректность. Если значение меньше 1 или больше 10, просится ввести размер заново. Дальше заполняем массив A элементами.
4. tests.asm - автоматические тесты, покрывающие все случаи.
   Тесты:
   1. Все элементы положительные.
   2. Все элементы отрицательные.
   3. Все элементы нули.
   4. Есть отрицательные, положительные элементы и нули.
   5. Есть только нули и положительные элементы.
   6. Есть только нули и отрицательные элементы.
   7. Есть только положительные и отрицательные элементы.
5. macros.asm - библиотека с макросами:
   1. get_size - получение размера массива.
   2. check_size - проверка размера массива на корректность.
   3. add_element - считывание элемента для массива.
   4. print_element - вывод элемента на экран.
   5. print_array - вывод элементов массива.
   6. add_positive - добавление положительных элементов массива A в массив B.
   7. add_zeros - добавление нулей из массива A в массив B.
   8. add_negative - добавление отрицательных элементов массива A в массив B.

Результаты автоматических тестов, покрывающих все случаи

```
MENU
Enter '0': input your array.
Enter other symbol: launch autotests.

Your choice: 3
1 2 3 4 5
1 2 3 4 5

-1 -2 -3 -4
-1 -2 -3 -4

0 0 0
0 0 0

-4 5 0 -5 52
5 52 0 -4 -5

0 1 234 0
1 234 0 0

0 -1 -234 0
0 0 -1 -234

-123 1 234 -5
1 234 -123 -5


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

Пример работы программы

```
MENU
Enter '0': input your array.
Enter other symbol: launch autotests.

Your choice: 0
Input the size of the array from 1 to 10: 7
Input element: -234
Input element: 435
Input element: 0
Input element: 456
Input element: 1
Input element: -54
Input element: -7
435 456 1 0 -234 -54 -7

If you want to finish program, enter '0', otherwise press any other number to restart: 4
Input the size of the array from 1 to 10: 3
Input element: 0
Input element: -123
Input element: 706
706 0 -123

If you want to finish program, enter '0', otherwise press any other number to restart: 0

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

Выполненные условия:

На 4-5

1. Есть решение на ассемблере. Есть ввод и вывод данных на экран.
2. Есть комментарии.
3. Использование подпрограмм без параметров и локальных переменных.
4. Отчёт с полным тестовым покрытием.
   На 6-7
5. Использование подпрогрмм с передачей аргументов через регистры по конвенции. В коде не понадобились такие аргументы, для всех подпрограмм они не нужны.
6. Сохранение локальных переменных в свободных регистрах.
7. Комментарии к функциям.
8. Изменения в отчёте. Решение сразу написано.
   На 8
9. Многократное использование подпрограмм.
10. Реализована дополнительная тестовая программа (tests.asm).
11. Добавлена информация в отчёт.
    На 9
12. Используются макросы.
    На 10
13. Программа разбита на несколько файлов.
14. Макросы выделены в отдельную автономную библиотеку.
15. Всё добавлено в отчёт.

main.asm

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

prompt_start:   .asciz "\nMENU\nEnter '0': input your array.\nEnter other symbol: launch autotests.\n\nYour
choice: "
prompt_next:    .asciz "If you want to finish program, enter '0', otherwise 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:
        # Output message for choice own array or autotests
        li      a7 4
        la      a0 prompt_start
        ecall
        # Read number of choice
        li      a7 5
        ecall
        beqz    a0 your_array
        j       to_tests
your_array:
        # Creating own array A
        call    user_array
        # Creating array B
        call    work

        # Message with an offer to continue
        li      a7 4
        la      a0 prompt_next
        ecall
        # Read number of choice
        li      a7 5
        ecall
        beqz    a0 end
        j       your_array
to_tests:
        # Launch autotests
```

```
        call    autotests
end:
        # Stop
        li      a7 10
        ecall
```

solution.asm

```
.text
work:
        # Adding positive elements, zeros, negative elements
        add_positive(array_A, array_B, n, t6)
        add_zeros(array_A, t6, n, t6)
        add_negative(array_A, t6, n)
        # Print array
        print_array(array_B, n)
        # Print new line
        li      a7 4
        la      a0 newline
        ecall
        ret
```

user.asm

```
.data
prompt_size:    .asciz "Input the size of the array from 1 to 10: "
prompt_er_size: .asciz "\nYou need input a number from 1 to 10. Try again!\n"
prompt_element: .asciz "Input element: "

.text
user_array:
size:
        # Getting array size
        get_size(t0)
        la      t1 n
        sw      t0 (t1)
        mv      a0 t0
        # Checking size
        check_size(t1)
        beqz    t1 size
make_array_A:
        # Creating array A
        la      t0 array_A
        li      t2 0
        lw      t3 n
add_el:
        # Adding element in array A
        beq     t2 t3 stop
        add_element(t1)
        sw      t1 (t0)
        addi    t0 t0 4
        addi    t2 t2 1
        j       add_el
stop:
        ret
```

tests.asm

```
.text
autotests:
        # Save return address
        addi    sp sp -4
        sw      ra (sp)

        # Call tests and print result
        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

        # Load return address
        lw      ra (sp)
        addi    sp sp 4
        ret

# test 1
store_test_1:
        la      t0 array_A
        li      t1 1
        sw      t1 (t0)
        li      t1 2
        sw      t1 4(t0)
        li      t1 3
        sw      t1 8(t0)
        li      t1 4
        sw      t1 12(t0)
        li      t1 5
        sw      t1 16(t0)
        la      t0 n
        li      t1 5
        sw      t1 (t0)
        ret

# test 2
store_test_2:
        la      t0 array_A
        li      t1 -1
        sw      t1 (t0)
        li      t1 -2
        sw      t1 4(t0)
        li      t1 -3
        sw      t1 8(t0)
        li      t1 -4
        sw      t1 12(t0)
        la      t0 n
        li      t1 4
        sw      t1 (t0)
        ret

# test 3
store_test_3:
        la      t0 array_A
        li      t1 0
        sw      t1 (t0)
        li      t1 0
        sw      t1 4(t0)
        li      t1 0
        sw      t1 8(t0)
        la      t0 n
        li      t1 3
        sw      t1 (t0)
        ret

# test 4
```

```
store_test_4:
        la      t0 array_A
        li      t1 -4
        sw      t1 (t0)
        li      t1 5
        sw      t1 4(t0)
        li      t1 0
        sw      t1 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
store_test_5:
        la      t0 array_A
        li      t1 0
        sw      t1 (t0)
        li      t1 1
        sw      t1 4(t0)
        li      t1 234
        sw      t1 8(t0)
        li      t1 0
        sw      t1 12(t0)
        la      t0 n
        li      t1 4
        sw      t1 (t0)
        ret

# test 6
store_test_6:
        la      t0 array_A
        li      t1 0
        sw      t1 (t0)
        li      t1 -1
        sw      t1 4(t0)
        li      t1 -234
        sw      t1 8(t0)
        li      t1 0
        sw      t1 12(t0)
        la      t0 n
        li      t1 4
        sw      t1 (t0)
        ret

# test 7
store_test_7:
        la      t0 array_A
        li      t1 -123
        sw      t1 (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
```

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

.macro add_positive(%A, %B, %size, %pos)
        # Set begging of array A and array B to register t0 and t1
        la      t0 %A
        la      t1 %B
        li      t2 0    # Counter of elements
        lw      t3 %size
next:
        beq     t2 t3 end
        # Load element from array A
        lw      t4 (t0)
        # Cheecking that this element is correct
```

```
        blez    t4 skip
        # Adding the element tot arrray B
        sw      t4 (t1)
        addi    t1 t1 4
skip:
        # Go to the next element
        addi    t0 t0 4
        addi    t2 t2 1
        j       next
end:
        mv      %pos t1
.end_macro

.macro add_zeros(%A, %B, %size, %pos)
        # Set begging of array A and array B to register t0 and t1.
        la      t0 %A
        mv      t1 %B
        li      t2 0    # Counter of elements
        lw      t3 %size
next:
        beq     t2 t3 end
        # Load element from array A
        lw      t4 (t0)
        # Cheecking that this element is correct
        bnez    t4 skip
        # Adding the element tot arrray B
        sw      t4 (t1)
        addi    t1 t1 4
skip:
        # Go to the next element
        addi    t0 t0 4
        addi    t2 t2 1
        j       next
end:
        mv      %pos t1
.end_macro

.macro add_negative(%A, %B, %size)
        # Set begging of array A and array B to register t0 and t1.
        la      t0 %A
        mv      t1 %B
        li      t2 0    # Counter of elements
        lw      t3 %size
next:
        beq     t2 t3 end
        # Load element from array A
        lw      t4 (t0)
        # Cheecking that this element is correct
        bgez    t4 skip
        # Adding the element tot arrray B
        sw      t4 (t1)
        addi    t1 t1 4
skip:
        # Go to the next element
        addi    t0 t0 4
        addi    t2 t2 1
        j       next
end:
.end_macro
```