НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»
ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Лабораторна робота № 1.2

з дисципліни

«Архітектура для програмістів»

Тема:

"ТРАНСЛЯЦІЯ МОВ ВИСОКОГО РІВНЯ У МОВИ НИЗЬКОГО РІВНЯ. Ч.2"

Виконав: студент III курсу
ФПМ групи КВ-94
Кувашов Я.Р.
Перевірив: Молчанов О.А.

Київ 2022

Загальне завдання

1. Реалізувати програму сортування масиву згідно із варіантом мовою Java.

2. Виконати трансляцію програми, написаної мовою Java, у байт-код Java за допомогою javac і javap (програми, що постачаються разом з пакетом openjdk) й встановити семантичну відповідність між командами мови Java та командами одержаного байт-коду Java, додававши коментарі з поясненням.

3. Виконати порівняльний аналіз відповідних семантичних частин програм, записаних мовою асемблера (лабораторна робота 1.1) та байт-кодом Java.

Завдання за варіантом 10

Відсортувати побічну діагональ масиву алгоритмом No2 методу обмінів («бульбашкове сортування» з використанням «прапорця») за незбільшенням.

Лістинг програми мовою Java

```
static void sort(int size, int Array[][])
{
    int tmp;
    int R = size - 1;
    boolean flag = true;
    while(flag == true)
    {
        flag = false;
    for (int i = 0; i < R; ++i)
      {
       if(Array[i+1][size-2-i]>Array[i][size-1-i])
                      {
                tmp = Array[i+1][size-2-i];
            Array[i+1][size-2-i] = Array[i][size-1-i];
                  Array[i][size-1-i] = tmp;
                   flag = true;
                            }

      }
          R--;
    }
}
```

Лістинг програми байт-кодом Java з поясненнями

```
static void sort(int, int[][]);
    descriptor: (I[[I)V
    flags: (0x0008) ACC_STATIC
// function starts
    Code:
      stack=5, locals=6, args_size=2
      StackMap locals:  int int[][]
      StackMap stack:
        start local 0 // int size
        start local 1 // int[][] Array
// int R = size - 1;
         0: iload_0    // load int value of  variable size to stack
         1: iconst_1   // push int constant 1 onto the operand stack
         2: isub       // size - 1
         3: istore_3   // store size-1 into local variable R
        start local 3  // int R
//boolean flag = true;
         4: iconst_1  // push int constant 1 onto the operand stack
         5: istore        4  // pull int value from stack and put to local
variable vith index 4 ( flag = 1)
        start local 4 // boolean flag
      StackMap locals:  int int[][] top int int
      StackMap stack:
//while (flag == 1)  loop starts
//while(flag == true)
         7: iload         4    // load int value of  variable flag to stack
         9: iconst_1          // push int constant 1 onto the operand stack
 //while (flag == 1)  loop condition
         10: if_icmpne     117  //  if_icmpne succeeds if and only if value1 ≠
                                                value2 ( if flag !=1) goto 117
// flag = false;
```

```
        13: iconst_0           // push int constant 0 onto the operand stack

          14: istore        4    // pull int value from stack and put to local
                                    variable vith index 4 ( flag = 0)

//for (int i = 0; i < R; ++i)

        16: iconst_0             // push int constant 0 onto the operand
stack

              17: istore        5      // pull int value from stack and put to
                                         local variable vith index 5 ( i = 0)

        start local 5 // int i

      StackMap locals:  int int[][] top int int int

      StackMap stack:

        19: iload         5      // load int value of  variable i to stack

        21: iload_3              // load int value of  variable R to stack

//main loop for( int i = 0;i<n-1;i++) condition

         22: if_icmpge     111    //if_icmpge succeeds if and only if value1 ≥
                                                value2  if i>=R goto 111

// main loop for( int i = 0;i<n-1;i++) starts

//if(Array[i+1][size-2-i]>Array[i][size-1-i])

        25: aload_1

        26: iload         5

        28: iconst_1

        29: iadd

        30: aaload

        31: iload_0

        32: iconst_2

        33: isub

        34: iload         5

        36: isub

        37: iaload

        38: aload_1

        39: iload         5

        41: aaload

        42: iload_0

        43: iconst_1
```

```
         44: isub
         45: iload         5
         47: isub
         48: iaload
         49: if_icmple     105
// if true branch start
//tmp = Array[i+1][size-2-i];
         52: aload_1
         53: iload         5
         55: iconst_1
         56: iadd
         57: aaload
         58: iload_0
         59: iconst_2
         60: isub
         61: iload         5
         63: isub
         64: iaload
         65: istore_2
         start local 2 // int tmp
//Array[i+1][size-2-i] = Array[i][size-1-i];
         66: aload_1
         67: iload         5
         69: iconst_1
         70: iadd
         71: aaload
         72: iload_0
         73: iconst_2
         74: isub
         75: iload         5
         77: isub
         78: aload_1
         79: iload         5
```

```
         81: aaload
         82: iload_0
         83: iconst_1
         84: isub
         85: iload         5
         87: isub
         88: iaload
         89: iastore
// Array[i][size-1-i] = tmp;
         90: aload_1
         91: iload         5
         93: aaload
         94: iload_0
         95: iconst_1
         96: isub
         97: iload         5
         99: isub
        100: iload_2
        101: iastore
//flag = true;
        102: iconst_1        // push int constant 1 onto the operand stack
            103: istore        4  // pull int value from stack and put to local
                                     variable vith index 4 ( flag = 1)
         end local 2 // int tmp
// if true branch end
        105: iinc          5, 1 // i++
        108: goto          19   // goto for loop head
//main loop for( int i = 0;i<n-1;i++) ends
         end local 5 // int i
//R--;
         111: iinc          3, -1 // increment  local variable vith index 3 -->
                                                                  R + (-1)
        114: goto          7     // goto while loop new iteration
//while (flag == 1)  loop ends
```

```
        117: return
         end local 4 // boolean flag
         end local 3 // int R
         end local 1 // int[][] Array
         end local 0 // int size
// function ends
```

Порівняльний аналіз

№	Код мовою C	Код мовою Java	Assembly language	Java Bytecode	Опис
1	_Bool flag = 1 ;	boolean flag = true;	mov BYTE PTR [rbp-25], 1	4: iconst_1 5: istore 4 start local 4	Визначення змінної flag і запис в неї значення 1.
2	while (flag == 1)	while (flag == true)	cmp BYTE PTR [rbp-25], 0 jne .L6	7: iload 4 9: iconst_1 10: if_icmpne 117	Перевірка істинності умови циклу
3	for(i = 0;i<R;i++)	for(int i=0;i<R;++i)	mov DWORD PTR [rbp-20], 0 // i=0 jmp .L3 .L4: add DWORD PTR [rbp-20], 1 // i++ .L3: mov edx, DWORD PTR [rbp-20] cmp edx, DWORD PTR [rbp-24] jl .L5	16: iconst_0 17: istore 5 start local 5 105: inc 5, 1 19: iload 5 21: iload_3 22: if_icmpge 111 111: iinc 3, -1	Реалізація циклу for Різниця в підході до обробки умови i<R В асм перевіряється умова i<R а в байт-коді умова i>=R
4	if(<cond>) statement	if(<cond>) statement	<cond> cmp ecx, edx jle .L4 <statement> .L4 …	<cond> if_icmple 105 <statement> 105: …	Реалізація умового переходу if
5	i++	i++	add DWORD PTR [rbp-20], 1	iinc 5,1	Інкремент змінної i
6	R--	R--	sub DWORD PTR [rbp-24], 1	iinc 3, -1	Декремент змінної R
7	i<R	i<R	mov edx, DWORD PTR [rbp-20] cmp edx, DWORD PTR [rbp-24] jl .L5	19: iload 5 21: iload_3 22: if_icmpge 111	Перевірка умови виходу з циклу for