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[][]);
//function body starts
       0: iload_0     // load int value vith index 0 (size)
       1: iconst_1    // push int constant with value 1
       2: isub       //  size-1
       3: istore_3   // int  R = size-1

       4: iconst_1  //   push int constant with value 1
       5: istore        4  // store constant 1 (flag = 1)
// while(flag == true)loop starts
       7: iload         4 // load int value vith index 4 (flag)
       9: iconst_1        // push int constant with value 1
// while(flag == true)exit condition starts
      10: if_icmpne     117 // Branch if  cmp succeeds  (if flag != 1) goto
117
// while(flag == true)exit condition ends

// while(flag == true)loop body starts
      13: iconst_0          //push int constant with value 0
      14: istore        4    // store constant 0 into int with index 4 (flag=
0)
//loop for (int i = 0; i < R; ++i) starts
      16: iconst_0           // push int constant with value 0
      17: istore        5     // store constant 0 into int with index 5 ( i =
0)
//loop for (int i = 0; i < R; ++i) exit condition starts
      19: iload         5     // load int value vith index 5 (int i)
      21: iload_3              // load int value vith index 3 (R)
      22: if_icmpge     111    // succeeds if and only if i≥R  goto 111
//loop for (int i = 0; i < R; ++i) exit condition ends

//loop for (int i = 0; i < R; ++i) body starts
```

```
// if check starts :if(Array[i+1][size-2-i]<=Array[i][size-1-i]
      25: aload_1             // Load reference from local variable
      26: iload         5      //   load int value vith index 5 (i)
      28: iconst_1            //   push int constant with value 1
      29: iadd               // i+1
      30: aaload              //  load Array[i+1]
      31: iload_0            // load int value vith index 0 (size)
      32: iconst_2          //   push int constant with value 2
      33: isub              // size-2
      34: iload         5    //   load int value vith index 5 (i)
      36: isub               // size-2-i
      37: iaload             // Load  Array[i+1][size-2-i]
      38: aload_1            // Load reference from local variable
      39: iload         5    //   load int value vith index 5 (i)
      41: aaload             //  load Array[i]
      42: iload_0              // load int value vith index 0 (size)
      43: iconst_1            //    push int constant with value 1
      44: isub                // size-1
      45: iload         5     //  load int value vith index 5 (i)
      47: isub                // size-1-i
      48: iaload                // Load Array[i][size-1-i]
      49: if_icmple     105  // if(Array[i+1][size-2-i]<=Array[i][size-1-i])
goto 105
// if check ends :if(Array[i+1][size-2-i]<=Array[i][size-1-i]


// if  true branch starts
      52: aload_1               // Load reference from local variable
      53: iload         5      // load int value vith index 5 (i)
      55: iconst_1               // push int constant with value 1
      56: iadd                  // i+1
      57: aaload                // load Array[i+1]
      58: iload_0           // load int value vith index 0 (size)
      59: iconst_2           // push int constant with value 2
```

```
60: isub                // size-2
61: iload         5      //  load int value with index 5 (i)
63: isub                 // size-2-i
64: iaload              //  load Array[i+1][ size-2-i]
65: istore_2            // (tmp = Array[i+1][ size-2-i] )

66: aload_1              // Load reference from local variable
67: iload         5      //  load int value with index 5 (i)
69: iconst_1             //  push int constant with value 1
70: iadd                // i+1
71: aaload              // load Array[i+1]
72: iload_0             // load int value with index 0 (size)
73: iconst_2            // push int constant with value 2
74: isub                // size-2
75: iload         5    //  load int value with index 5 (i)
77: isub                // size-2-i
78: aload_1             // Load Array[i+1][size-2-i]
79: iload         5    // load int value with index 5 (i)
81: aaload              // load Array[i]
82: iload_0             // load int value with index 0 (size)
83: iconst_1             //  push int constant with value 1
84: isub                // size-1
85: iload         5      //  load int value with index 5 (i)
87: isub                 // size-1-i
88: iaload               //  load Array[i][ size-1-i]
89: iastore              // Array[i+1][size-2-i] = Array[i][size-1-i]

90: aload_1             //  load Array
91: iload         5    //  load int value with index 5 (i)
93: aaload              // load Array[i]
94: iload_0             // load int value with index 0 (size)
95: iconst_1            //  push int constant with value 1
96: isub                // size-1
```

```
      97: iload         5    //    load int value with index 5 (i)
      99: isub                // size-1-i
     100: iload_2             // load tmp
     101: iastore            // Array[i][size-1-i] = tmp

     102: iconst_1           //  push int constant with value 1
     103: istore        4  // store constant 1 (flag= 1)
// if  true branch ends
//loop for (int i = 0; i < R; ++i) body ends
     105: iinc          5, 1  // i++
     108: goto          19
//loop for (int i = 0; i < R; ++i) ends
     111: iinc          3, -1 // R--
// while(flag == true)loop body ends
     114: goto          7     // goto while(flag == true) cond.loop
// while(flag == true)loop ends
     117: return
// function body 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`	Визначення змінної flag і запис в неї значення 1. Значення true/false та тип даних bool/_Bool/Boolean є макросом на значення 1/0 та int відповідно. Через необхідність роботи з неявними параметрами, що беруться зі стеку, байт-код налічує дві інструкції для аналогічного коду, записаного мовою асемблера

2	`while (condition) {`	`while (condition) {`	`cmp BYTE PTR [rbp-25], 0`
			`jne .L6`
				`7: iload 4`
				`9: iconst_1`
				`10:`
					Перевірка істинності умови циклу

	do_something(); }	do_something(); }		if_icmpne 117	
3	for (; condition;) { do_something(); }	for (; condition;) { do_something(); }	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 19:iload 5 21: iload_3 22: if_icmpge 111	Реалізація циклу for Різниця в підході до обробки умови i<R В asm перевіряється умова i<R а в JBC байт-коді умова i>=R Також в asm Обробка умови винесена в окремий блок .L3, звідки при виконанні умови йде перехід в тіло циклу. В JBC порівняння виконується по ходу програми, а перехід за межі тіла йде при невиконанні умови порівняння
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
8	A[i]	Array[i]	mov rdx, QWORD PTR [rbp-64]	90: aload_1 91:iload 5 93: aaload	отримання значення з визначеної комірки масиву (за

			lea rsi, [rcx+rdx] mov edx, DWORD PTR [rbp-52]		індексом). В асемблерному коді відбувається приведення типів даних, і вирахування адреси