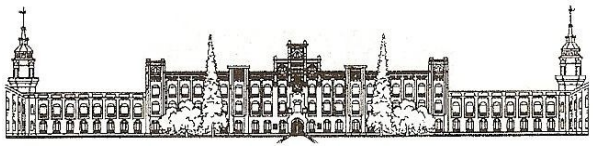

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА СИСТЕМНОГО ПРОГРАМУВАННЯ ТА СПЕЦІАЛІЗОВАНИХ КОМП’ЮТЕРНИХ СИСТЕМ

Лабораторна робота №2

з дисципліни «Архітектура для програмістів»

Виконав студент групи: КВ-22

ПІБ: Крутогуз Максим Ігорович

Перевірив: Молчанов О. А.

Київ 2025

Трансляція мов високого рівня у мови низького рівня. Частина 2

Мета лабораторної роботи наступна:

- ознайомитись із роботою сучасних трансляторів на прикладі трансляції невеликої програми, що містить різні конструкції керування потоком виконання і написаної мовою програмування високого рівня, у код цієї програми мовою низького рівня;
- навчитись виокремлювати синтаксичні/семантичні конструкції програми, записаної мовою високого рівня, у відповідній їй програмі, записаній мовою низького рівня, на прикладі мови високого рівня Java

Варіант #12

Задано двовимірний масив (матрицю) цілих чисел A[m][n]. Відсортувати окремо кожен рядок масиву алгоритмом No3 методу вставки (з лінійним пошуком справа з використанням бар’єру) за не- збільшенням.

Код 1 — Лістинг мовою Java

```java
public static void sortArray(int[][] arr, int m, int n) {
    for (int i = 0; i < m; i++) {
        for (int j = 2; j < n; j++) {
            arr[i][0] = arr[i][j];
            int k = j;

            while (arr[i][0] < arr[i][k - 1]) {
                arr[i][k] = arr[i][k - 1];
                k--;
            }

            arr[i][k] = arr[i][0];
        }
    }
}
```

Код 2 — Лістинг байт-кодом Java

```
// Function begin
public static void sortArray(int[][], int, int);
   // i loop start
```

```
 0: iconst_0
 1: istore_3
 // i loop condition start
 2: iload_3
 3: iload_1
 4: if_icmpge     90 // for (int i = 0; i < m; i++) {
 // i loop condition end
 // i loop body start
 // j loop start
 7: iconst_2
 8: istore        4
// j loop condition start
10: iload         4
12: iload_2
13: if_icmpge     84 // for (int j = 2; j < n; j++) {
// j loop condition end
// j loop body start
16: aload_0
17: iload_3
18: aaload
19: iconst_0
20: aload_0
21: iload_3
22: aaload
23: iload         4
25: iaload
26: iastore // arr[i][0] = arr[i][j]
27: iload         4
29: istore        5 // int k = j
// while (condition) loop start
31: aload_0
32: iload_3
33: aaload
34: iconst_0
35: iaload
36: aload_0
37: iload_3
38: aaload
39: iload         5
41: iconst_1
42: isub
43: iaload
44: if_icmpge     67 // while (arr[i][0] < arr[i][k - 1]) {
// while condition loop end
// while loop body start
47: aload_0
48: iload_3
49: aaload
50: iload         5
52: aload_0
53: iload_3
54: aaload
55: iload         5
57: iconst_1
58: isub
59: iaload
60: iastore // arr[i][k] = arr[i][k - 1]
61: iinc          5, -1
64: goto          31 //   k--
// while loop (body) end
67: aload_0
```

```
    68: iload_3
    69: aaload
    70: iload          5
    72: aload_0
    73: iload_3
    74: aaload
    75: iconst_0
    76: iaload
    77: iastore // arr[i][k] = arr[i][0]
    // j loop body end
    78: iinc           4, 1
    81: goto           10 // for (int j = 2; j < n; j++) {
    // j loop end
    // i loop body end
    84: iinc           3, 1
    87: goto           2 // for (int i = 0; i < m; i++) {
    // i loop end
    90: return

// Function end
```

Таблиця 1 — Порівняльний аналіз асемблерного коду і байт-коду Java

№	Код мовою C	Код мовою Java	Асемблерний код	Байт-код Java	Опис

1	`void sortArray(int *arr, int m, int n) {}`	`public static void sortArray(int[][] arr, int m, int n) {}`	`push rbp`
			`mov rbp, rsp`
			`mov QWORD PTR [rbp-24], rdi`
			`mov DWORD PTR [rbp-28], esi`
			`mov DWORD PTR [rbp-32], edx`
			
			`...`
			
			`nop`
			`nop`
			`pop rbp`
			`ret`
				...
				`90: return`
					Ініціалізує початкові значення в стек. В java, ми не працюємо явним чином із регістрами, тому ініціалізація явна непотрібна

2	`for (int i = 0; i < m; i++) {}`	`for (int i = 0; i < m; i++) {}`	`mov DWORD PTR [rbp-4], 0`
			`jmp .L9`
			`.L14`
			
			`...`
			
			`add DWORD PTR [rbp-4], 1`
			`.L9:`
			`mov eax, DWORD PTR [rbp-4]`
				`0: iconst_0`
				`1: istore_3`
				`2: iload_3`
				`3: iload_1`
				`4: if_icmpge 90`
				
				...
				
				`84: iinc 3, 1`
				`87: goto 2`
					Цикл із змінною і спочатку ініціалізується змінна початковим значенням, далі в залежності від асемблерового коду чи байт коду відбувається перевірка умови,

			cmp eax, DWORD PTR [rbp-28] jl .L14		а в кінці інкремент.
3	for (int j = 2; j < n; j++) {}	for (int j = 2; j < n; j++) {}	mov DWORD PTR [rbp-8], 2 jmp .L10 .L13 ... add DWORD PTR [rbp-8], 1 .L10: mov eax, DWORD PTR [rbp-8] cmp eax, DWORD PTR [rbp-32] jl .L13	7: iconst_2 8: istore 4 10: iload 4 12: iload_2 13: if_icmpge 84 ... 78: iinc 4, 1 81: goto 10 84:	Схоже як і в попередньому прикладі

4	arr[i*n + 0] = arr[i*n + j];	arr[i][0] = arr[i][j];	mov eax, DWORD PTR [rbp-4]
			imul eax, DWORD PTR [rbp-32]
			mov edx, eax
			mov eax, DWORD PTR [rbp-8]
			add eax, edx
			cdqe
			lea rdx, [0+rax*4]
			mov rax, QWORD PTR [rbp-24]
			lea rcx, [rdx+rax]
			mov eax, DWORD PTR [rbp-4]
			imul eax, DWORD PTR [rbp-32]
			cdqe
			lea rdx, [0+rax*4]
			mov rax, QWORD PTR [rbp-24]
			add rdx, rax
			mov eax, DWORD PTR [rcx]
			mov DWORD PTR [rdx], eax
				16: aload_0
				17: iload_3
				18: aaload
				19: iconst_0
				20: aload_0
				21: iload_3
				22: aaload
				23: iload 4
				25: iaload
				26: iastore
					В асемблерському коду ми обраховуємо адреси двох елементів масиву. Далі записуємо по адресі одного елементу значення іншого елементу масиву. В байт коді ми поступово отримуємо адресу та індекс елементу і таким чином отримуємо адресу або значення елементу/масиву і повторюючи схожий процес в кінці маємо в java стеці три елементи: перший це адрес масиву по якому

					буде звернення і другий це індекс по якому буде запис, а третій це значення, яке буде записано.
5	`int k = j;`	`int k = j;`	`mov eax, DWORD PTR [rbp-8]` `mov DWORD PTR [rbp-12], eax`	`27: iload 4` `29: istore 5`	В асемблеровому коді маємо капіювання локальної змінної через регістр. А в байт коді, маємо копіювання та завантаження через віртуальний стек.

6	`while (arr[i*n + 0] < arr[i*n + k - 1]) {}`	`while (arr[i][0] < arr[i][k - 1]) {}`	`jmp .L11`
			`...`
			`.L11:`
			`mov eax, DWORD PTR [rbp-4]`
			`imul eax, DWORD PTR [rbp-32]`
			`cdqe`
			`lea rdx, [0+rax*4]`
			`mov rax, QWORD PTR [rbp-24]`
			`add rax, rdx`
			`mov edx, DWORD PTR [rax]`
			`mov eax, DWORD PTR [rbp-4]`
			`imul eax, DWORD PTR [rbp-32]`
			`mov ecx, eax`
			`mov eax, DWORD PTR [rbp-12]`
			`add eax, ecx`
			`cdqe`
			`sal rax, 2`
			`lea rcx,`
				`31: aload_0`
				`32: iload_3`
				`33: aaload`
				`34: iconst_0`
				`35: iaload`
				`36: aload_0`
				`37: iload_3`
				`38: aaload`
				`39: iload 5`
				`41: iconst_1`
				`42: isub`
				`43: iaload`
				`44: if_icmpge 67`
					Подібно до четвертого прикладу, але замість присвоєння, отримуємо значення в два регістра в асемблері та порівнюємо. Також відмінність в тому, що робимо віднімання на одиницю
					В байт коді подібно, але умова перевернута та переходи відбуваються по іншому

			[rax-4]
			mov rax, QWORD PTR [rbp-24]
			add rax, rcx
			mov eax, DWORD PTR [rax]
			cmp edx, eax
			jl .L12		

7	arr[i* n + k] = arr[i*n + k - 1];	arr[i][k] = arr[i][k - 1];	L12:
			mov eax, DWORD PTR [rbp-4]
			imul eax, DWORD PTR [rbp-32]
			mov edx, eax
			mov eax, DWORD PTR [rbp-12]
			add eax, edx
			cdqe
			sal rax, 2
			lea rdx, [rax-4]
			mov rax, QWORD PTR [rbp-24]
			lea rcx, [rdx+rax]
			mov eax, DWORD PTR [rbp-4]
			imul eax, DWORD PTR [rbp-32]
			mov edx, eax
			mov eax, DWORD PTR [rbp-12]
			add eax, edx
			cdqe
			lea rdx, [0+rax*4]
			mov rax, QWORD PTR [rbp-24]
			add rdx, rax
			mov eax, DWORD PTR [rcx]
			mov DWORD PTR [rdx], eax
				47: aload_0
				48: iload_3
				49: aaload
				50: iload 5
				52: aload_0
				53: iload_3
				54: aaload
				55: iload 5
				57: iconst_1
				58: isub
				59: iaload
				60: iastore
					Подібно до прикладу 4, але ще із відніманням як в прикладі 6.

	k--;	k--;	sub DWORD PTR [rbp-12], 1	61: iinc 5, -1 64: goto 31	Просто віднімання одиниці

	arr[i * n + k] = arr[i * n + 0];	arr[i][k] = arr[i][0];	mov eax, DWORD PTR [rbp-4]
			imul eax, DWORD PTR [rbp-32]
			cdqe
			lea rdx, [0+rax*4]
			mov rax, QWORD PTR [rbp-24]
			lea rcx, [rdx+rax]
			mov eax, DWORD PTR [rbp-4]
			imul eax, DWORD PTR [rbp-32]
			mov edx, eax
			mov eax, DWORD PTR [rbp-12]
			add eax, edx
			cdqe
			lea rdx, [0+rax*4]
			mov rax, QWORD PTR [rbp-24]
			add rdx, rax
			mov eax, DWORD PTR [rcx]
			mov DWORD PTR [rdx], eax
				67: aload_0
				68: iload_3
				69: aaload
				70: iload 5
				72: aload_0
				73: iload_3
				74: aaload
				75: iconst_0
				76: iaload
				77: iastore
					Схожий до прикладу 7 та 4.

Посилання на [репозиторій]