Теорія мов програмування

Лабораторна робота 3. Асемблер JCoCo

3.1 Код на рис. 4.2 (див. лекцію 4) трохи марнотратний, що часто трапляється при складанні програми, написаної мовою вищого рівня. Оптимізуйте код на рис. 4.2, щоб він містив менше інструкцій.

3.2 Не торкаючись коду, який порівнює два значення, збірку на рис. 5.2 (див. лекцію 5) можна оптимізувати до видалення принаймні трьох інструкцій. Перепишіть код, щоб видалити принаймні три інструкції з цього коду. Трохи більше роботи та п’ять інструкцій можна буде видалити.

3.3 Перепишіть код на рис. 5.3 (див. лекцію 5), щоб він виконувався з однаковим результатом, використовуючи `POP_JUMP_IF_TRUE` замість переходу, якщо є помилкова інструкція. Обов’язково оптимізуйте свій код, коли пишете його, щоб не було зайвих інструкцій.

3.4 Напишіть коротку програму, яка перевіряє використання інструкції `BREAK_LOOP`. Вам не потрібно писати цикл `while`, щоб перевірити це. Просто напишіть код, який використовує `BREAK_LOOP` і надрукує щось на екран, щоб переконатися, що він працював.

3.5 Напишіть коротку програму, яка перевіряє створення винятку, його створення та друк обробленого винятку. Напишіть це як програму JCoCo без використання розбирача (дізасемблера).

3.6 Зазвичай, якщо ви хочете додавати числа до Python разом, наприклад `5` і `6`, ви пишете `5 + 6`. Це відповідає використанню інструкції `BINARY_ADD` у JCoCo, яка, у свою чергу, викликає магічний метод `__add__` із викликом методу `5.__add__(6)`. Напишіть коротку програму JCoCo, де ви додасте два цілих числа разом, не використовуючи інструкцію `BINARY_ADD`. Надрукуйте результат на екран.

3.7 Напишіть програму JCoCo, яка отримує рядок від користувача та перебирає символи рядка, друкуючи їх на екран.

3.8 Програма на рис. 6.2 прекрасно працювала б без комірки. Змінна `x` може посилатися безпосередньо на `3` у функціях `f` та `g` без будь-яких наслідків. Тим не менше, змінна комірки потрібна в деяких випадках. Чи можете ви привести приклад, коли змінна комірки є абсолютно необхідною?

3.9 Намалюйте зображення стеку часу виконання безпосередньо перед тим, як буде виконана інструкція в рядку 11 на рис. 6.4. Використовуйте рис. 6.3 як

підказку щодо того, як ви малюєте цю картинку. Не забудьте включити код, значення `n` та значення ПК.

Виконані завдання надішліть на пошту викладачеві.

Контрольні запитання

1. Чим відрізняються віртуальна машина Python та JCoCo? Назвіть три відмінності між двома реалізаціями.
2. Що таке розбирач (дізасемблер)?
3. Що таке асемблер?
4. Що таке кадр стека? Де вони зберігаються? Що входить у кадр стека?
5. Яке призначення стека блоків і де він зберігається?
6. Яка мета лічильника програм (Program Counter)?
7. Назвіть інструкцію, яка відповідає за створення об’єкта списку, та опишіть, як вона працює.
8. Опишіть виконання інструкцій `STORE_FAST` та `LOAD_FAST`.
9. Як JCoCo може прочитати рядок введення з клавіатури?
10. У чому різниця між розібраною програмою Python та зібраною програмою JCoCo? Наведіть короткий приклад і вкажіть на відмінності.
11. Коли в JCoCo реалізований цикл Python while, яка остання інструкція циклу і яке його призначення?
12. Що спільного у обробці винятків та циклах у реалізації JCoCo?
13. Що таке ледача оцінка і чому це важливо для Python та JCoCo?
14. Що таке закриття та навіщо потрібне закриття?
15. Як створюється екземпляр класу в JCoCo? Які вказівки необхідно виконати для створення об’єктів?
16. Напишіть клас, використовуючи JCoCo, і створіть деякі екземпляри класу.