ЛР № 4. Робота з симулятором машини Ноймана. Дослідження архітектури системи команд.

Мета: зрозуміти принципи виконання архітектури системи команд на симуляторі машини Ноймана, зрозуміти і дослідити виконання інструкції.

Завдання: розширити архітектуру систему команд симулятора машини Ноймана, скласти програму на асемблері з розширеним набором команд, перетворити її у машинні коди, запустити симулятор, увести до нього коди машинних, проаналізувати і пояснити отримані результати, довести коректність роботи розширеного набору команд, скласти звіт з виконання лабораторних досліджень та захистити його.

Теоретичні відомості

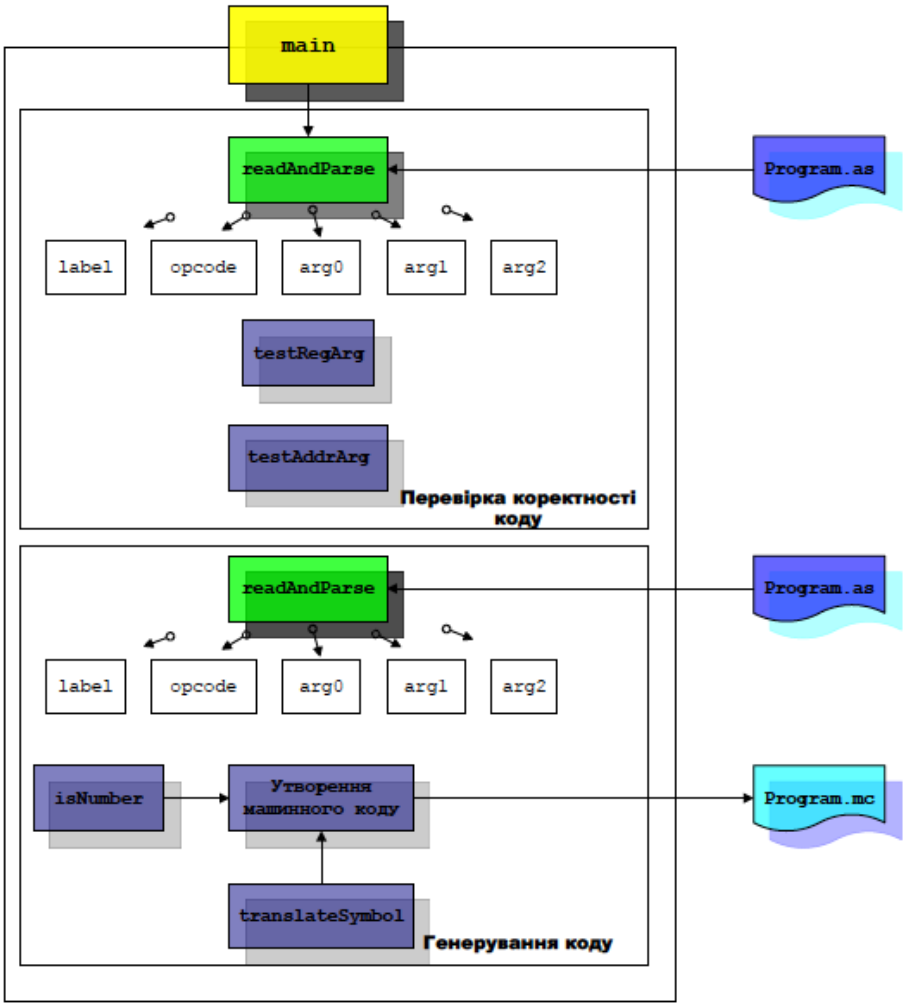

Рис. 4.1. Схема роботи асемблера

Загальна схема роботи асемблера (рис. 1) складається з 2 проходів. На першому проході асемблер перевіряє коректність синтаксису команд. На другому виконується генерування відповідних машинних команд, тобто числового представлення асемблерної команди.

Функція *readAndParse* виконує зчитування рядку асемблерної програми і декодування на відповідні поля: мітка, код операції, операнди. Отримана таким чином і декодована інструкція перевіряється на коректність: існування команди, відповідна кількість аргументів, існування міток та т. п.

Функція *testRegArg* перевіряє коректність використання назви регістра.

Функція *testAddrArg* перевіряє коректність використання адреси.

Функція *labelArray* перетворює відповідну мітку у адресу.

Program.as та *program.mc* – відповідно вхідний та вихідний файли.

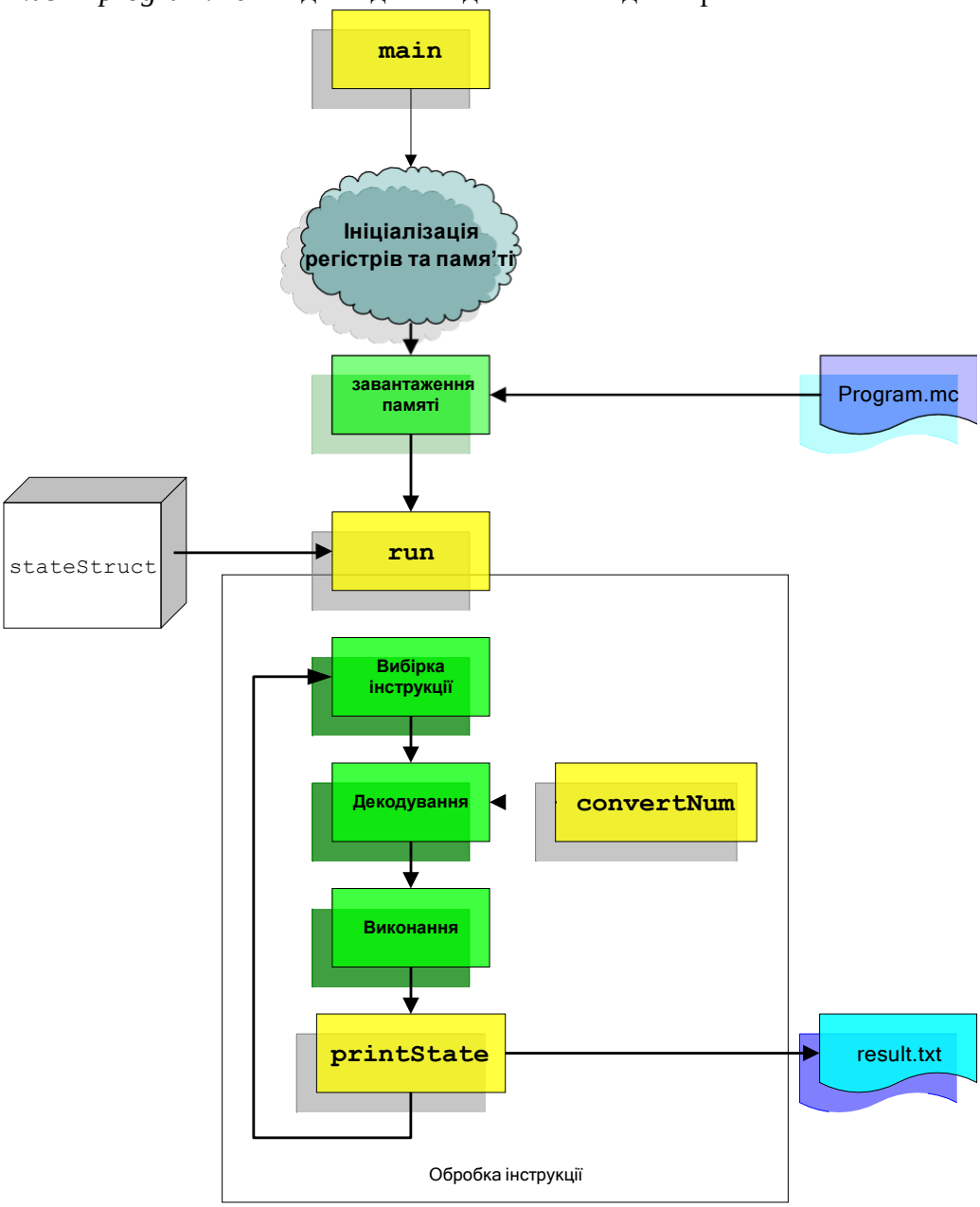

Рис. 4.2 Функціональна схема симулятора.

Симулятор починає свою роботу ініціалізацією пам’яті та регістрів 0 значеннями (рис. 2.). Наступним кроком відбувається заванаження програми у машинних кодах в пам’ять. Далі відбувається покрокове виконання інструкцій та вивід стану на зовнішній пристрій (чи на екран консолі чи у файл).

У stateStruct зберігається стан машини – значення регістрів, пам’яті та програмний лічільник. *stateStruct*

Функція *Run* виконує обробку інструкцій з пам’яті, функція *printState* виводить поточний стан машини, а функція *convertNum* виконує перетворення числа у доповняльний код.

Хід виконання роботи:

1. Відкрити вихідні файли з вихідними кодами (assol.c ssol.c)
2. Відкомпілювати дані вихідні коди у окремих проектах.
3. Дослідити алгоритм роботи асемблерної та симуляційної програми.
4. Замінити інструкцію noop власною згідно індивідуального варіанту.
5. Перетворити асемблер ний код у машинний.
6. Запустити симулятор з отриманим у п.5 машинним кодом.
7. Проаналізувати хід виконання машинних інструкцій, перевірити правильність результатів.
8. Скласти звіт по результатам виконання програми.

Склад звіту:

1. Титульний аркуш (№ лабораторної роботи, тема, назва предмету).
2. Мета
3. Фрагменти коду в які були внесені зміни в ході виконання роботи.
4. Лістінг тестової програми.
5. Результат виконання. Приводити початковий і кінцевий стани машини повністю , статистику, проміжні стани лише які регістри чи пам’ять, які зазнали змін.
6. Вмсновок.

Завдання

Скласти програму на асемблерній мові симулятора з новою командою згідно варіанту.Персональні варіанти завдань знайдете у табличці з варіантами.