НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Факультет ПИИКТ

Системы искусственного интеллекта

Лабораторная работа № 2

Выполнил студент

Куприянов А.А

Группа № P33113

г. Санкт-Петербург

2020

Исходный представлен в gitlab и github репозиториях

https://github.com/AppLoidx/Assembly/tree/master/dictionary

https://gitlab.se.ifmo.ru/apploid/low-level-labs/-/tree/master/dictionary

Реализация связанного списка через макросы ассемблера

```
%define last_elem 0

%macro colon 2
%%last_elem: dq last_elem
db %1, 0
my_super_label_ %+ %2:

%define last_elem %%last_elem
%endmacro
```

Вывод

Эта лабораторная работа познакомила нас с макросами (в частности в NASM), но и у других программ принцип работы похожий.

В свою очередь макросы – это поистине мощные инструменты призванные сократить не только объем кода, который нужно написать программисту, но и часто допускаемые ошибки (все мы знаем насколько может быть черевато дублирование) я и не говорю об удобочитаемости кода в целом.

Тем не менее, несмотря на богатый функционал макросов, кто-то может сказать, что вместо них можно использовать вызов функции (например, мы использовали их в первой лабораторной работе). Но суждение будет ошибочным, потому что макросы выполняются не в рантайме, а просто выполняется так называемое *“макрорасширение”* – вместо макроса использованном в коде во время компиляции вставляется сам ассемблерный код. Да, это повышает сложность разработки самих компиляторов, но эта лабораторная работа показала, что сложность оправдана.

Не менее важной частью этой лабораторной работы было изучение этапов компиляции. Я, например, раньше не понимал (или вовсе понимал неправильно) про линковку. Казалось бы, очень неудобно вызывать сначала команду nasm, а потом еще и ld (из-за этого приходится писать скрипты для компиляции)

Но это решение является действительно хорошим в случае с большими проектами. Оказывается, что линкование происходит намного быстрее, чем компиляция ассемблерного файла в объектный. Если представить ситуацию, что у нас большой проект, состоящий из нескольких сотен отдельных файлов, то при хоть малейшем изменении одной, нам придеться компилировать все файлы, если бы не было отдельного процесса линкования.

А сейчас мы должны будем перекомпилировать лишь измененный файл, а дальше вызвать процесс линкования, который обходится дешевле, чем компиляция всего проекта.

Дополнительно, лабораторная работа включала в себя использование программы для сброки make. В первой лабораторной рабое, я написал себе shell-скрипт для сборки. Но как только появилась необходимость линковать несколько файлов писать такой скрипт стало намного тяжелее.

Тут помогает утилита **make**, которая сама по определенной логике (через транзитивные зависимости) определяет какая команда должна выполняться первее.

Удобно, например, вызывать отдельные команды или вызвать сразу несколько:
make all clean, который все скомпилирует и очистит все промежуточные файлы (при правильной конфигурации **Makefile**, конечно).

Системы сборки играют важную роль в разработке приложения и даже на таком низкоуровневом языке программирования как ассемблер есть сборщик (хотя, на самом деле, не конкретно под ассемблерные языки)