МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Гжельский государственный университет»** (ГГУ)

Колледж ГГУ

Специальность 09.02.07 Информационные системы и программирование

**Реферат**

по предмету «Моделирование и анализ ПО»

на тему «Дизассемблирование»

ВЫПОЛНИЛА:

Студентка группы ИСП-О-17

Пигарева Е. А.

ПРОВЕРИЛА:

Прокуронова А. Ю.

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

п. Электроизолятор

1. г.

**Дизассе́мблер**—  [транлятор](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D1%81%D0%BB%D1%8F%D1%82%D0%BE%D1%80), преобразующий  [машинный код](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA), [объектный файл](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D1%8B%D0%B9_%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8C) или библиотечные модули в текст [программы](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0) на [языке ассемблера](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%B0%D1%81%D1%81%D0%B5%D0%BC%D0%B1%D0%BB%D0%B5%D1%80%D0%B0).

По режиму работы с пользователем делятся на автоматические и интерактивные. Примером автоматических дизассемблеров может служить **Sourcer**. Такие дизассемблеры генерируют готовый листинг, который можно затем править в текстовом редакторе. Пример интерактивного — [IDA](https://ru.wikipedia.org/wiki/IDA). Он позволяет изменять правила дизассемблирования и является весьма удобным инструментом для [исследования программ](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC).

Дизассемблеры бывают однопроходные и многопроходные. Основная трудность при работе дизассемблера — отличить данные от машинного кода, поэтому на первых проходах автоматически или интерактивно собирается информация о границах процедур и функций, а на последнем проходе формируется итоговый листинг. Интерактивность позволяет улучшить этот процесс, так как просматривая дамп дизассемблируемой области памяти, программист может сразу выделить строковые константы, дать содержательные имена известным точкам входа, прокомментировать разобранные им фрагменты программы. Обычно однопроходный дизассемблер (как и построчный ассемблер) является составной частью [отладчика](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BB%D0%B0%D0%B4%D1%87%D0%B8%D0%BA).

Чаще всего дизассемблер используют для анализа [программы](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0) (или её части), [исходный текст](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D1%82%D0%B5%D0%BA%D1%81%D1%82) которой неизвестен — с целью модификации, копирования или взлома. Реже — для поиска ошибок ([багов](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B3)) в [программах](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0" \o "Компьютерная программа)и [компиляторах](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80), а также для анализа и оптимизации создаваемого [компилятором](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80) машинного кода.

**Базовый алгоритм дизассемблирования**

* Шаг 1. Первым шагом в процессе дизассемблирования является идентификация кодового сегмента. Так как команды обычно смешаны с данными, то дизассемблеру необходимо их разграничить.
* Шаг 2. Получив адрес первой команды, необходимо прочитать значение, содержащееся по этому адресу (или смещению в файле) и выполнить табличное преобразование двоичного кода операции в соответствую ему мнемонику языка ассемблера.
* Шаг 3. Как только команда была обнаружена и декодирована, ее ассемблерный эквивалент может быть добавлен к результирующему листингу. После этого необходимо выбрать одну из разновидностей синтаксиса языка ассемблера.
* Шаг 4. Далее необходимо перейти к следующей команде и повторить предыдущие шаги до тех пор, пока каждая команда файла не будет дизассемблирована.

**Алгоритм линейной развертки**

Алгоритм линейной развертки использует крайне прямолинейный подход при выборе очередной команды для дизассемблирования: где завершается код одной команды - начинается код новой. В результате, наиболее сложной задачей становится определение первой команды. Дизассемблирование начинается с первого байта в сегменте кода и последовательно продвигается, обрабатывая команды одну за другой, пока не будет достигнут конец сегмента. При этом не производится попыток понять логику передач управления в программе посредством распознавания команд перехода таких, например, как условия.

Главное преимущество алгоритма линейной развертки состоит в полном покрытии кодового сегмента. Одним из основных недостатков является невозможность распознать данные, если они совмещены с кодом. Это очевидно из листинга, демонстрирующего результат дизассемблирования функции при помощи данного алгоритма. Эта функция содержит конструкцию switсh. Компилятор принял решение реализовать switch как таблицу переходов. Более того, компилятор предпочел разместить таблицу переходов внутри самой функции. Конструкция jump по адресу 401250 ссылается на таблицу адресов начиная с 410257. К сожалению, дизассемблер рассматривает ее как набор команд и неверно генерирует соответствующее представление на языке ассемблера.

При работе с исполняемым кодом или байт-кодом, созданным на некоторых языках высокого уровня (например, java) имеется возможность восстановить не только текст на языке ассемблера, но даже и структуру классов программы, а если при компиляции исполняемого файла не была отключена отладочная информация — то и исходный текст программы. Для исключения таких возможностей используется [обфускация](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%84%D1%83%D1%81%D0%BA%D0%B0%D1%86%D0%B8%D1%8F" \o "Обфускация).

Примеры программ-дизассемблеров

* [IDA](https://ru.wikipedia.org/wiki/IDA)
* [Sourcer](https://ru.wikipedia.org/w/index.php?title=Sourcer&action=edit&redlink=1)
* [Hiew](https://ru.wikipedia.org/wiki/Hiew)
* [Beye](https://ru.wikipedia.org/wiki/Beye)
* [HT editor](https://ru.wikipedia.org/w/index.php?title=HT_editor&action=edit&redlink=1)
* [Hacker Disassembler Engine](https://ru.wikipedia.org/w/index.php?title=Hacker_Disassembler_Engine&action=edit&redlink=1)
* [CADt](https://ru.wikipedia.org/w/index.php?title=CADt&action=edit&redlink=1)
* [Objdump](https://ru.wikipedia.org/wiki/Objdump)
* [Radare2](https://ru.wikipedia.org/wiki/Radare2)
* [Ghidra](https://ru.wikipedia.org/w/index.php?title=Ghidra&action=edit&redlink=1)[[en]](https://en.wikipedia.org/wiki/Ghidra)

Но существует всего лишь один дизассемблер, пригодный для по-настоящему профессиональной работы — IDA Pro.

Этот дизассемблер имеет консольную и графическую (рис. 1.8) версии. IDA Pro воспринимает огромное количество форматов файлов и множество типов процессоров, легко справляясь с байт-кодом виртуальных машин Java и .NET, поддерживает макросы, плагины и скрипты, содержит интегрированный отладчик, работает под MS-DOS, Windows, Linux (рис. 1.9) и обладает уникальной способностью распознавать имена стандартных библиотечных функций по их сигнатурам.

Существует несколько версий IDA Pro — бесплатная (freeware), стандартная (standard) и расширенная (advanced).

Основное достоинство IDA Pro состоит в том, что это — интерактивный дизассемблер, то есть интеллектуальный инструмент, позволяющий работать с двоичным файлом, мыслить и творить,

а не тупой автомат, заглатывающий исследуемую программу и выплевывающий "готовый" дизассемблированный листинг, в котором все дизассемблировано неправильно.

**Консольная версия IDA Pro.**

Существуют следующие префиксы:  
Первые шесть изменяют сегментный регистр, используемый командой при обращении к ячейке памяти.

• 0x26 – префикс замены сегмента ES.  
• 0x2E – префикс замены сегмента CS.  
• 0x36 – префикс замены сегмента SS.  
• 0x3E – префикс замены сегмента DS.  
• 0x64 – префикс замены сегмента FS.  
• 0x65 – префикс замены сегмента GS.  
• 0x0F – префикс дополнительных.  
• 0x66 – префикс переопределения размера операнда (к примеру, вместо регистра будет использоваться).  
• 0x67 – префикс переопределения размера адреса.  
• 0x9B – префикс ожидания (WAIT).  
• 0xF0 – префикс блокировки (LOCK с его помощью реализуется синхронизация многопоточных приложений).  
• 0xF2 – префикс повторенья команды REPNZ – работа с последовательностями байт (строками).  
• 0xF3 – префикс повторенья команды REP – работа с последовательностями байт (строками).

Каждый из этих префиксов меняет семантику и (или) структуру машинной инструкции (например, ее длину или выбор мнемоники).

**Опкоды команд.**   
**Опкод** команды иногда один, иногда вмести с префиксом (ами) однозначно определяет мнемонику (название) команды. Команд много. И при усложнении современных микропроцессоров их количество не уменьшается – новые команды появляются, а устаревшие не исчезают (обратная совместимость). Список опкодов и команд ассоциированных с ними, как правило, можно скачать на официальных сайтах производителей микропроцессоров.

**Байт Mod\_R/M состоит из следующих полей:**• Mod – первые два бита (значение от 0 до 3)   
• R/M – следующие три бита (значение от 0 до 7)  
• Value of ModR/M – следующие три бита (значение от 0 до 7)