# ЛЕКЦИЯ 1. ИНСТРУМЕНТАЛЬНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

Инструментальное программное обеспечение — программное обеспечение, предназначенное для использования в ходе проектирования, разработки и сопровождения программ, в отличие от прикладного и системного программного обеспечения.

# 

# 1.1. СИСТЕМЫ ПРОГРАММИРОВАНИЯ

К этой категории относятся программы, предназначенные для разработки программного обеспечения:

* ассемблеры — компьютерные программы, осуществляющие преобразование программы в форме исходного текста на языке ассемблера в машинные команды в виде объектного кода;
* трансляторы — программы или технические средства, выполняющие трансляцию программы;
* компоновщики (редакторы связей) — программы, которые производят компоновку — принимают на вход один или несколько объектных модулей и собирают по ним исполняемый модуль;
* препроцессоры исходных текстов — это компьютерные программы, принимающие данные на входе и выдающие данные, предназначенные для входа другой программы, например, такой, как компилятор;
* отладчики (debugger) являются модулями среды разработки или отдельными приложениями, предназначенными для поиска ошибок в программе;
* текстовые редакторы — компьютерные программы, предназначенные для создания и изменения текстовых файлов, а также их просмотра на экране, вывода на печать, поиска фрагментов текста и т. п.;
* специализированные редакторы исходных текстов — текстовые редакторы для создания и редактирования исходного кода программ. Специализированный редактор исходных текстов может быть отдельным приложением, или быть встроен в интегрированную среду разработки (IDE);
* библиотеки подпрограмм — сборники подпрограмм или объектов, используемых для разработки программного обеспечения;
* редакторы графического интерфейса;
* средства автоматизации разработки программ (CASE-средства).

Перечисленные инструменты могут входить в состав интегрированных сред разработки.

# 

# 1.2. ВИДЫ ИНСТРУМЕНТАЛЬНОГО ПО

1. Интегрированные среды разработки
2. SDK
3. Компиляторы и кросс-компиляторы
4. Интерпретаторы
5. Линковщики (компоновщики)
6. Парсеры и генераторы парсеров
7. Ассемблеры
8. Отладчики
9. Профилировщики
10. Генераторы документации
11. Средства анализа покрытия кода
12. Средства непрерывной интеграции
13. Средства автоматизированного тестирования
14. Системы управления версиями
15. Системы управления проектами
16. Системы отслеживания ошибок

## 

## 1.2.1. Интегрированная среда разработки

Интегрированная среда разработки, ИСP (англ. Integrated development environment — IDE), также единая среда разработки, ЕСР — комплекс программных средств, используемый программистами для разработки программного обеспечения (ПО).

Среда разработки включает в себя:

* текстовый редактор,
* компилятор и/или интерпретатор,
* средства автоматизации сборки,
* отладчик.

Иногда содержит также средства для интеграции с системами управления версиями и разнообразные инструменты для упрощения конструирования графического интерфейса пользователя. Многие современные среды разработки также включают браузер классов, инспектор объектов и диаграмму иерархии классов — для использования при объектно-ориентированной разработке ПО.

ИСР обычно предназначены для нескольких языков программирования — такие как IntelliJ IDEA, NetBeans, Eclipse, Qt Creator, Geany, Embarcadero RAD Studio, Code::Blocks, Xcode или Microsoft Visual Studio, но есть и IDE для одного определённого языка программирования — как, например, Visual Basic, Delphi, Dev-C++.

Частный случай ИСР — среды визуальной разработки, которые включают в себя возможность наглядного редактирования интерфейса программы.

Использование ИСР для разработки программного обеспечения является прямой противоположностью способу, в котором используются несвязанные инструменты, такие как текстовый редактор, компилятор, и т. п.

Интегрированные среды разработки были созданы для того, чтобы максимизировать производительность программиста благодаря тесно связанным компонентам с простыми пользовательскими интерфейсами. Это позволяет разработчику сделать меньше действий для переключения различных режимов, в отличие от дискретных программ разработки.

Однако так как ИСР является сложным программным комплексом, то среда разработки сможет качественно ускорить процесс разработки ПО лишь после специального обучения. Для уменьшения барьера вхождения многие достаточно интерактивны, а для облегчения перехода с одной на другую интерфейс у одного производителя максимально близок, вплоть до использования одной ИСР.

ИСР обычно представляет собой единственную программу, в которой проводится вся разработка. Она, как правило, содержит много функций для создания, изменения, компилирования, развертывания и отладки программного обеспечения.

Цель интегрированной среды заключается в том, чтобы объединить различные утилиты в одном модуле, который позволяет абстрагироваться от выполнения вспомогательных задач, тем самым позволяя программисту сосредоточиться на решении собственно алгоритмической задачи и избежать потерь времени при выполнении типичных технических действий (например, вызове компилятора).

Таким образом, повышается производительность труда разработчика. Также считается, что тесная интеграция задач разработки может далее повысить производительность за счет возможности введения дополнительных функций на промежуточных этапах работы. Например, ИСР позволяет проанализировать код и тем самым обеспечить мгновенную обратную связь и уведомить о синтаксических ошибках.

Большинство современных ИСР являются графическими. Но первые ИСР использовались ещё до того, как стали широко применяться операционные системы с графическим интерфейсом — они были основаны на текстовом интерфейсе с использованием функциональных и горячих клавиш для вызова различных функций (например, Turbo Pascal, созданный фирмой Borland).

## 1.2.2. SDK

SDK (от англ. software development kit) — набор средств разработки, который позволяет специалистам по программному обеспечению создавать приложения для определённого пакета программ, программного обеспечения базовых средств разработки, аппаратной платформы, компьютерной системы, игровых консолей, операционных систем и прочих платформ.

Программист, как правило, получает SDK непосредственно от разработчика целевой технологии или системы. Часто SDK распространяется через Интернет. Многие SDK распространяются бесплатно для того, чтобы побудить разработчиков использовать данную технологию или платформу.

Поставщики SDK иногда подменяют слово «software» в словосочетании «software development kit» на более точное слово. Например, Microsoft и Apple предоставляют Driver Development Kit (DDK) для разработки драйверов устройств, PalmSource называет свой инструментарий для разработки PalmOS Development Kit (PDK), а Oracle — Java Development Kit (JDK).

Примеры SDK:

* Android SDK.
* Windows Phone SDK.
* Adobe Flex.
* DirectX.
* iPhone SDK.
* Java Development Kit.
* Opera Devices SDK.
* Source SDK.
* bada SDK.
* CryEngine 3 SDK
* X-Ray SDK

## 

## 1.2.3. Компилятор

Компилятор — программа или техническое средство, выполняющее компиляцию.

Компиляция — трансляция программы, составленной на исходном языке высокого уровня, в эквивалентную программу на низкоуровневом языке, близком машинному коду (абсолютный код, объектный модуль, иногда на язык ассемблера).

Входной информацией для компилятора (исходный код) является описание алгоритма или программа на предметно-ориентированном языке, а на выходе компилятора — эквивалентное описание алгоритма на машинно-ориентированном языке (объектный код).

Компилировать — проводить трансляцию машинной программы с предметно-ориентированного языка на машинно-ориентированный язык.

### Виды компиляторов:

* Векторизующий. Транслирует исходный код в машинный код компьютеров, оснащенных векторным процессором.
* Гибкий. Сконструирован по модульному принципу, управляется таблицами и запрограммирован на языке высокого уровня или реализован с помощью компилятора компиляторов.
* Диалоговый.
* Инкрементальный. Повторно транслирует фрагменты программы и дополнения к ней без перекомпиляции всей программы.
* Интерпретирующий (пошаговый). Последовательно выполняет независимую компиляцию каждого отдельного оператора (команды) исходной программы.
* Компилятор компиляторов. Транслятор, воспринимающий формальное описание языка программирования и генерирующий компилятор для этого языка.
* Отладочный. Устраняет отдельные виды синтаксических ошибок.
* Резидентный. Постоянно находится в оперативной памяти и доступен для повторного использования многими задачами.
* Самокомпилируемый. Написан на том же языке, с которого осуществляется трансляция.
* Универсальный. Основан на формальном описании синтаксиса и семантики входного языка. Составными частями такого компилятора являются: ядро, синтаксический и семантический загрузчики.

Виды компиляции:

* Пакетная. Компиляция нескольких исходных модулей в одном пункте задания.
* Построчная. Машинный код порождается и затем исполняется для каждой завершённой грамматической конструкции языка. Внешне воспринимается как интерпретация, но устройство иное.
* Условная. Компиляция, при которой транслируемый текст зависит от условий, заданных в исходной программе директивами компилятора. Так, в зависимости от значения некоторой константы, можно включать или выключать трансляцию части текста программы.

Процесс компиляции состоит из следующих этапов:

* Лексический анализ. На этом этапе последовательность символов исходного файла преобразуется в последовательность лексем.
* Синтаксический (грамматический) анализ. Последовательность лексем преобразуется в дерево разбора.
* Семантический анализ. Дерево разбора обрабатывается с целью установления его семантики (смысла) — например, привязка идентификаторов к их декларациям, типам, проверка совместимости, определение типов выражений и т. д. Результат обычно называется «промежуточным представлением/кодом», и может быть дополненным деревом разбора, новым деревом, абстрактным набором команд или чем-то ещё, удобным для дальнейшей обработки.
* Оптимизация. Выполняется удаление излишних конструкций и упрощение кода с сохранением его смысла. Оптимизация может быть на разных уровнях и этапах — например, над промежуточным кодом или над конечным машинным кодом.
* Генерация кода. Из промежуточного представления порождается код на целевом языке.

В конкретных реализациях компиляторов эти этапы могут быть разделены или, наоборот, совмещены в том или ином виде.

### 

### 1.2.3.1. Генерация кода

#### 

#### Генерация машинного кода

Большинство компиляторов переводит программу с некоторого высокоуровневого языка программирования в машинный код, который может быть непосредственно выполнен процессором. Как правило, этот код также ориентирован на исполнение в среде конкретной операционной системы, поскольку использует предоставляемые ею возможности (системные вызовы, библиотеки функций). Архитектура (набор программно-аппаратных средств), для которой производится компиляция, называется целевой машиной.

Результат компиляции — исполнимый модуль — обладает максимальной возможной производительностью, однако привязан к определённой операционной системе и процессору (и не будет работать на других).

Для каждой целевой машины (IBM, Apple, Sun и т. д.) и каждой операционной системы или семейства операционных систем, работающих на целевой машине, требуется написание своего компилятора. Существуют также так называемые кросс-компиляторы, позволяющие на одной машине и в среде одной ОС генерировать код, предназначенный для выполнения на другой целевой машине и/или в среде другой ОС.

Кроме того, компиляторы могут оптимизировать код под разные модели из одного семейства процессоров (путём поддержки специфичных для этих моделей особенностей или расширений наборов инструкций). Например, код, скомпилированный под процессоры семейства Pentium, может учитывать особенности распараллеливания инструкций и использовать их специфичные расширения — MMX, SSE и т. п.

Некоторые компиляторы переводят программу с языка высокого уровня не прямо в машинный код, а на язык ассемблера (примером может служить PureBasic, транслирующий бейсик-код в ассемблер FASM).

Это делается для упрощения части компилятора, отвечающей за кодогенерацию, и повышения его переносимости (задача окончательной генерации кода и привязки его к требуемой целевой платформе перекладывается на ассемблер), либо для возможности контроля и исправления результата компиляции программистом.

Генерация байт-кода

Результатом работы компилятора может быть программа на специально созданном низкоуровневом языке, подлежащем интерпретации виртуальной машиной. Такой язык называется псевдокодом или байт-кодом.

Как правило, он не является машинным кодом какого-либо компьютера и программы на нём могут исполняться на различных архитектурах, где имеется соответствующая виртуальная машина, но в некоторых случаях создаются аппаратные платформы, напрямую поддерживающие псевдокод какого-либо языка.

Например, псевдокод языка Java называется байт-кодом Java и выполняется в Java Virtual Machine, для его прямого исполнения была создана спецификация процессора picoJava. Для платформы .NET Framework псевдокод называется Common Intermediate Language (CIL), а среда исполнения — Common Language Runtime (CLR).

Некоторые реализации интерпретируемых языков высокого уровня (например, Perl) используют байт-код для оптимизации исполнения: затратные этапы синтаксического анализа и преобразование текста программы в байт-код выполняются один раз при загрузке, затем соответствующий код может многократно использоваться без промежуточных этапов.

Динамическая компиляция

Из-за необходимости интерпретации байт-код выполняется значительно медленнее машинного кода сравнимой функциональности, однако он более переносим (не зависит от операционной системы и модели процессора). Чтобы ускорить выполнение байт-кода, используется динамическая компиляция, когда виртуальная машина транслирует псевдокод в машинный код непосредственно перед его первым исполнением (и при повторных обращениях к коду исполняется уже скомпилированный вариант).

Наиболее популярной разновидностью динамической компиляции является JIT. Другой разновидностью является инкрементальная компиляция.

CIL-код также компилируется в код целевой машины JIT-компилятором, а библиотеки .NET Framework компилируются заранее.

Декомпиляция

Существуют программы, которые решают обратную задачу — перевод программы с низкоуровневого языка на высокоуровневый. Этот процесс называют декомпиляцией, а такие программы — декомпиляторами. Но поскольку компиляция — это процесс с потерями, точно восстановить исходный код, скажем, на C++, в общем случае невозможно. Более эффективно декомпилируются программы в байт-кодах — например, существует довольно надёжный декомпилятор для Flash.

Разновидностью декомпилирования является дизассемблирование машинного кода в код на языке ассемблера, который почти всегда выполняется успешно (при этом сложность может представлять самомодифицирующийся код или код, в котором собственно код и данные не разделены). Связано это с тем, что между кодами машинных команд и командами ассемблера имеется практически взаимно-однозначное соответствие.

Раздельная компиляция

Раздельная компиляция (англ. separate compilation) — трансляция частей программы по отдельности с последующим объединением их компоновщиком в единый загрузочный модуль.

Исторически особенностью компилятора, отраженной в его названии (англ. compile — собирать вместе, составлять), являлось то, что он производил как трансляцию, так и компоновку, при этом компилятор мог порождать сразу машинный код.

Однако позже, с ростом сложности и размера программ (и увеличением времени, затрачиваемого на перекомпиляцию), возникла необходимость разделять программы на части и выделять библиотеки, которые можно компилировать независимо друг от друга.

При трансляции каждой части программы компилятор порождает объектный модуль, содержащий дополнительную информацию, которая потом, при компоновке частей в исполнимый модуль, используется для связывания и разрешения ссылок между частями.

Появление раздельной компиляции и выделение компоновки как отдельной стадии произошло значительно позже создания компиляторов. В связи с этим вместо термина «компилятор» иногда используют термин «транслятор» как его синоним: либо в старой литературе, либо когда хотят подчеркнуть его способность переводить программу в машинный код (и наоборот, используют термин «компилятор» для подчёркивания способности собирать из многих файлов один).

## 1.2.4. Интерпретаторы

Интерпретатор — программа (разновидность транслятора), выполняющая интерпретацию. Интерпретация — построчный анализ, обработка и выполнение исходного кода программы или запроса (в отличие от компиляции, где весь текст программы, перед запуском, анализируется и транслируется в байт-код, без её выполнения).

### 

### Типы интерпретаторов

Простой интерпретатор анализирует и тут же выполняет (собственно интерпретация) программу покомандно (или построчно), по мере поступления её исходного кода на вход интерпретатора. Достоинством такого подхода является мгновенная реакция. Недостаток — такой интерпретатор обнаруживает ошибки в тексте программы только при попытке выполнения команды (или строки) с ошибкой.

Интерпретатор компилирующего типа — это система из компилятора, переводящего исходный код программы в промежуточное представление, например, в байт-код или p-код, и собственно интерпретатора, который выполняет полученный промежуточный код (так называемая виртуальная машина).

Достоинством таких систем является большее быстродействие выполнения программ (за счёт выноса анализа исходного кода в отдельный, разовый проход, и минимизации этого анализа в интерпретаторе). Недостатки — большее требование к ресурсам и требование на корректность исходного кода. Применяется в таких языках, как Java, PHP, Tcl, Perl, REXX (сохраняется результат парсинга исходного кода), а также в различных СУБД.

В случае разделения интерпретатора компилирующего типа на компоненты получаются компилятор языка и простой интерпретатор с минимизированным анализом исходного кода. Причем исходный код для такого интерпретатора не обязательно должен иметь текстовый формат или быть байт-кодом, который понимает только данный интерпретатор, это может быть машинный код какой-то существующей аппаратной платформы. К примеру, виртуальные машины вроде QEMU, Bochs, VMware включают в себя интерпретаторы машинного кода процессоров семейства x86.

Некоторые интерпретаторы (например, для языков Лисп, Scheme, Python, Бейсик и других) могут работать в режиме диалога или так называемого цикла чтения-вычисления-печати (англ. read-eval-print loop, REPL). В таком режиме интерпретатор считывает законченную конструкцию языка (например, s-expression в языке Лисп), выполняет её, печатает результаты, после чего переходит к ожиданию ввода пользователем следующей конструкции.

Уникальным является язык Forth, который способен работать как в режиме интерпретации, так и компиляции входных данных, позволяя переключаться между этими режимами в произвольный момент, как во время трансляции исходного кода, так и во время работы программ.

Следует также отметить, что режимы интерпретации можно найти не только в программном, но и аппаратном обеспечении. Так, многие микропроцессоры интерпретируют машинный код с помощью встроенных микропрограмм, а процессоры семейства x86, начиная с Pentium (например, на архитектуре Intel P6), во время исполнения машинного кода предварительно транслируют его во внутренний формат (в последовательность микроопераций).

Алгоритм работы простого интерпретатора:

* прочитать инструкцию;
* проанализировать инструкцию и определить соответствующие действия;  
  выполнить соответствующие действия;
* если не достигнуто условие завершения программы, прочитать следующую инструкцию и перейти к пункту 2.

Достоинства и недостатки интерпретаторов

Достоинства: большая переносимость интерпретируемых программ — программа будет работать на любой платформе, на которой есть соответствующий интерпретатор. Как правило, более совершенные и наглядные средства диагностики ошибок в исходных кодах. Меньшие размеры кода по сравнению с машинным кодом, полученным после обычных компиляторов.  
 Недостатки: интерпретируемая программа не может выполняться отдельно без программы-интерпретатора. Сам интерпретатор при этом может быть очень компактным. Интерпретируемая программа выполняется медленнее, поскольку промежуточный анализ исходного кода и планирование его выполнения требуют дополнительного времени в сравнении с непосредственным исполнением машинного кода, в который мог бы быть скомпилирован исходный код. Практически отсутствует оптимизация кода, что приводит к дополнительным потерям в скорости работы интерпретируемых программ.

## 

## 1.2.5. Компоновщик

Компоновщик (также редактор связей, от англ. link editor, linker) — инструментальная программа, которая производит компоновку («линковку»): принимает на вход один или несколько объектных модулей и собирает по ним исполнимый модуль.

Общая схема процесса линковки: из объектных файлов и статических библиотек собираются исполняемые файлы или новые статические или динамические библиотеки.

Для связывания модулей компоновщик использует таблицы символов, созданные компилятором в каждом из объектных модулей. Эти таблицы могут содержать символы следующих типов:

* Определённые или экспортируемые имена — функции и переменные, определённые в данном модуле и предоставляемые для использования другим модулям;
* Неопределённые или импортируемые имена — функции и переменные, на которые ссылается модуль, но не определяет их внутри себя;
* Локальные — могут использоваться внутри объектного файла для упрощения процесса настройки адресов.

Для большинства компиляторов, один объектный файл является результатом компиляции одного файла с исходным кодом. Если программа собирается из нескольких объектных файлов, компоновщик собирает эти файлы в единый исполнимый модуль, вычисляя и подставляя адреса вместо символов, в течение времени компоновки (статическая компоновка) или во время исполнения (динамическая компоновка).

Компоновщик может извлекать объектные файлы из специальных коллекций, называемых библиотеками. Если не все символы, на которые ссылаются пользовательские объектные файлы, определены, то компоновщик ищет их определения в библиотеках, которые пользователь подал ему на вход.

Обычно, одна или несколько системных библиотек используются компоновщиком по умолчанию. Когда объектный файл, в котором содержится определение какого-либо искомого символа, найден, компоновщик может включить его (файл) в исполнимый модуль (в случае статической компоновки) или отложить это до момента запуска программы (в случае динамической компоновки).

Работа компоновщика заключается в том, чтобы в каждом модуле определить и связать ссылки на неопределённые имена. Для каждого импортируемого имени находится его определение в других модулях, упоминание имени заменяется на его адрес.

Компоновщик обычно не выполняет проверку типов и количества параметров процедур и функций. Если надо объединить объектные модули программ, написанные на языках со строгой типизацией, то необходимые проверки должны быть выполнены дополнительной утилитой перед запуском редактора связей.

## 

## 1.2.6. Парсер

Парсер (англиц. parser; от parse – анализ, разбор) или синтаксический анализатор — часть программы, преобразующей входные данные (как правило, текст) в структурированный формат. Парсер выполняет синтаксический анализ текста.

Каждый транслятор (компилятор или интерпретатор) имеет встроенный парсер. Парсер выполняет преобразование текста, записанного на каком-либо языке программирования во внутреннее представление, удобное для дальнейшей работы транслятора.

Парсеры применяются в разработке компьютерных игр при работе с текстовыми файлами, хранящими 3D графику.

Парсеры применяются при разборе баз данных, сохранённых в текстовых форматах (таких, как CSV, XML и т. п.).

Наиболее часто встречающиеся виды парсеров:

* очередь классифицированных лексем;
* абстрактное дерево;
* иерархические структуры;
* таблицы данных;

Виды парсеров по числу чтений входных данных:

* однопроходные;
* многопроходные.

Исходный код парсера может быть:

* написан программистами;
* сгенерирован специализированными утилитами (например, GNU Bison, Yacc).

1.2.7. Ассемблеры

Ассемблер (от англ. assembler — сборщик) — транслятор исходного текста программы, написанной на языке ассемблера, в программу на машинном языке.

Как и сам язык, ассемблеры, как правило, специфичны для конкретной архитектуры, операционной системы и варианта синтаксиса языка. Вместе с тем существуют мультиплатформенные или вовсе универсальные (точнее, ограниченно-универсальные, потому что на языке низкого уровня нельзя написать аппаратно-независимые программы) ассемблеры, которые могут работать на разных платформах и операционных системах. Среди последних можно также выделить группу кросс-ассемблеров, способных собирать машинный код и исполняемые модули (файлы) для других архитектур и операционных систем.

Ассемблирование может быть не первым и не последним этапом на пути получения исполнимого модуля программы. Так, многие компиляторы с языков программирования высокого уровня выдают результат в виде программы на языке ассемблера, которую в дальнейшем обрабатывает ассемблер.

Также результатом ассемблирования может быть не исполняемый, а объектный модуль, содержащий разрозненные блоки машинного кода и данных программы, из которого (или из нескольких объектных модулей) в дальнейшем с помощью редактора связей может быть получен исполнимый файл.

Наиболее известными ассемблерами для операционной системы DOS являлись Borland Turbo Assembler (TASM), Microsoft Macro Assembler (MASM) и Watcom Assembler (WASM). Также в своё время был популярен простой ассемблер A86.

При появлении операционной системы Windows появилось расширение TASM, именуемое TASM 5+ (неофициальный пакет, созданный человеком с ником !tE’), позволившее создавать программы для выполнения в среде Windows. Последняя известная версия TASM — 5.3, поддерживающая инструкции MMX, на данный момент включена в Turbo C++ Explorer. Но официально развитие программы полностью остановлено. Последнее развитие компилятор получил благодаря современной среде разработки TASM Visual. Среда неофициальная, но с её помощью работа с компилятором многократно упрощается.

Microsoft поддерживает свой продукт под названием Microsoft Macro Assembler. Она продолжает развиваться и по сей день, последние версии включены в наборы DDK. Но версия программы, направленная на создание программ для DOS, не развивается. Кроме того, Стивен Хатчессон создал пакет для программирования на MASM под названием «MASM32».

В состав операционной системы Linux входит пакет binutils, включающий в себя ассемблер gas (GNU Assembler), использующий AT&T-синтаксис, в отличие от большинства других популярных ассемблеров, которые используют Intel-синтаксис (поддерживается с версии 2.10).

Также существует открытый проект ассемблера, версии которого доступны под различные операционные системы, и который позволяет получать объектные файлы для этих систем. Называется этот ассемблер NASM (Netwide Assembler).

Yasm — это переписанная с нуля версия NASM под лицензией BSD (с некоторыми исключениями). flat assembler (fasm) — молодой ассемблер под модифицированной для запрета перелицензирования (в том числе под GNU GPL) BSD-лицензией. Есть версии для KolibriOS, Linux, DOS и Windows; использует Intel-синтаксис и поддерживает инструкции x86-64.

Кроме того, многие фирмы-разработчики программного обеспечения, такие, как IAR или Keil, представили свои варианты ассемблеров. В ряде случаев применение этих ассемблеров оказывается более эффективным благодаря удобному набору директив и наличию среды программирования, объединяющей в себе профессиональный ассемблер и язык программирования Си, отладчик и менеджер программных проектов.

Пример программы на языке ASM-51 для микроконтроллера AT89S52:

mov SCON, #50h

mov TH1, #0FDh

orl TMOD, #20h

setb TR1

again:

clr RI

jnb RI, $

mov A, SBUF

jnb RI, $

clr TI

mov SBUF, A

jnb TI, $

sjmp again

Данная программа отправляет обратно полученный символ, через последовательный порт UART.

Процесс трансляции программы на языке ассемблера в объектный код принято называть ассемблированием. В отличие от компилирования, ассемблирование — более или менее однозначный и обратимый процесс. В языке ассемблера каждой мнемонике соответствует одна машинная инструкция, в то время как в языках программирования высокого уровня за каждым выражением может скрываться большое количество различных инструкций. В принципе, это деление достаточно условно, поэтому иногда трансляцию ассемблерных программ также называют компиляцией.