

Машинный код (также употребляются термины **собственный код**, или **платформенно-ориентированный код**, или **родной код**, или **нативный код** — от англ. *native code*) — система команд (язык) конкретной вычислительной машины (машинный язык), который интерпретируется непосредственно микропроцессором или микропрограммами данной вычислительной машины.

Каждая модель процессора имеет свой собственный машинный язык, хотя во многих моделях эти наборы команд сильно перекрываются. Говорят, что процессор *A* *совместим* с процессором *B*, если процессор *A* полностью «понимает» машинный код процессора *B*. Если процессор *A* знает несколько команд, которых не понимает процессор *B*, то *B* несовместим с *A*.

«Слова» машинного языка называются **машинными инструкциями**. Каждая из них описывает элементарное действие, выполняемое процессором, такое как «переслать байт из памяти в регистр». Программа — это просто длинный список инструкций, выполняемых процессором. Раньше процессоры просто выполняли инструкции одну за другой, но новые суперскалярные процессоры способны выполнять несколько инструкций за раз. Прямой поток выполнения команд может быть изменён инструкцией перехода, которая переносит выполнение на инструкцию с заданным адресом. Инструкция перехода может быть условной, выполняющей переход только при соблюдении некоторого условия.

Также инструкции бывают постоянной длины (у RISC, MISC-архитектур) и диапазонной (у CISC-архитектур; например, для архитектуры x86 команда имеет длину от 8 до 120 битов).

Система команд — соглашение о предоставляемых архитектурой средствах программирования, а именно: определённых типах данных, инструкций, системы регистров, методов адресации, моделей памяти, способов обработки прерываний и исключений, методов ввода и вывода.

Система команд представляется спецификацией соответствия (микро)команд наборам кодов (микро)операций, выполняемых при вызове команды, определяемых (микро)архитектурой системы. (При этом, на системах с различной (микро)архитектурой может быть реализована одна и та же система команд. Например, Intel Pentium и AMD Athlon имеют почти идентичные версии системы команд x86, но имеют радикально различный внутренний дизайн.)

Базовыми командами являются, как правило, следующие:

- арифметические, например «сложения» и «вычитания»;
- битовые, например «логическое и», «логическое или» и «логическое не»;
- присваивание данных, например «переместить», «загрузить», «выгрузить»;
- ввода-вывода, для обмена данными с внешними устройствами;
- управляющие инструкции, например «переход», «условный переход», «вызов подпрограммы», «возврат из подпрограммы».

Оптимальными в различных ситуациях являются разные способы построения системы команд.

- Если объединить наиболее часто используемую последовательность микроопераций под одной микрокомандой, то надо будет обеспечивать меньше

микрокоманд. Такое построение системы команд носит название CISC (Complex Instruction Set Computing), в распоряжении имеется небольшое число составных команд.

- С другой стороны, это объединение уменьшает гибкость системы команд. Вариант с наибольшей гибкостью - наличие множества близких к элементарным операциям команд. Это RISC (Reduced Instruction Set Computing), в распоряжении имеются усечённые, простые команды.
- Еще большую гибкость системы команд можно получить используя MISC подход, построенный на уменьшении количества команд до минимального и упрощении вычислительного устройства обработки этих команд.

Язык ассемблера

Язык ассемблера — тип языка программирования низкого уровня, представляющий собой формат записи машинных команд, удобный для восприятия человеком. Часто для краткости его называют просто ассемблером, что не верно.

Содержание

- 1 Содержание языка
- 2 Достоинства и недостатки
 - 2.1 Достоинства языка ассемблера
 - 2.2 Недостатки языка ассемблера
- 3 Применение
 - 3.1 Связывание ассемблерного кода с другими языками
- 4 Синтаксис
 - 4.1 Инструкции
 - 4.2 Директивы
 - 4.3 Пример программы
- 5 Происхождение и критика термина «язык ассемблера»
- 6 См. также
- 7 Ссылки
- 8 Литература

Содержание языка

Команды языка ассемблера один в один соответствуют командам процессора и, фактически, представляют собой удобную символьную форму записи (*мнемокод*) команд и их аргументов. Также язык ассемблера обеспечивает базовые программные абстракции: связывание частей программы и данных через метки с символьными именами (при *ассемблировании*)^[1] для каждой метки высчитывается адрес, после чего каждое вхождение метки заменяется на этот адрес) и *директивы*^[2].





Директивы ассемблера позволяют включать в программу блоки данных (описанные явно или считанные из файла); повторить определённый фрагмент указанное число раз; компилировать фрагмент по условию; задавать адрес исполнения фрагмента, отличный от адреса расположения в памяти^[уточнить!]; менять значения меток в процессе компиляции; использовать макроопределения с параметрами и др.

Каждая модель процессора, в принципе, имеет свой набор команд и соответствующий ему язык (или диалект) ассемблера.




Имеются процессоры, реализующие базис своих высокоуровневых языков (Forth, Lisp), где ассемблер имеет минимальные отличия от базового языка. В отличие, например, от существующих, ассемблер процессора Java не совместим синтаксически с своим высокоуровневым языком.

Достоинства и недостатки

Достоинства языка ассемблера

-  Минимальное количество избыточного кода, то есть использование меньшего количества команд и обращений в память, позволяет увеличить скорость и уменьшить размер программы.
-  Обеспечение полной совместимости и максимального использования возможностей нужной платформы: использование специальных инструкций и технических особенностей данной платформы.
-  При программировании на ассемблере становятся доступными специальные возможности: непосредственный доступ к аппаратуре, портам ввода-вывода и особым регистрам процессора, а также возможность написания самомодифицирующегося кода (то есть метапрограммирование, причём без необходимости программного интерпретатора).
-  Последние технологии безопасности, внедряемые в операционные системы, не позволяют делать самомодифицирующегося кода, так как исключают одновременную возможность исполнения инструкций и запись в одном и том же участке памяти (технология *W^X* в *BSD*-системах, *DEP* в *Windows*).

Недостатки языка ассемблера

-  Большие объёмы кода и большое число дополнительных мелких задач, что приводит к тому, что код становится очень сложно читать и понимать, а следовательно усложняется отладка и доработка программы, а также трудность реализации парадигм программирования и любых других соглашений, что приводит к сложности совместной разработки.
-  Меньшее количество доступных библиотек, их малая совместимость между собой.
-  Непереносимость на другие платформы (кроме двоично совместимых).

Применение

Напрямую вытекает из достоинств и недостатков.

Поскольку большие программы на ассемблере писать крайне неудобно, их пишут на языках высокого уровня. На ассемблере же пишут небольшие фрагменты или модули, для которых критически важны:

- быстроедействие (драйверы);
- размер кода (загрузочные сектора, программное обеспечение для микроконтроллеров и процессоров с ограниченными ресурсами, вирусы, программные защиты);

- специальные возможности: работа напрямую с аппаратурой или машинным кодом, то есть загрузчики операционных систем, драйверы, вирусы, системы защиты.

Связывание ассемблерного кода с другими языками

Поскольку на ассемблере чаще всего пишут лишь фрагменты программы, их необходимо связывать с остальными частями на других языках. Это достигается 2 основными способами:

- **На этапе компиляции** — вставка в программу ассемблерных фрагментов (англ. *inline assembler*) специальными директивами языка, в том числе написание процедур на языке ассемблера. Способ удобен для несложных преобразований данных, но полноценного ассемблерного кода, с данными и подпрограммами, включая подпрограммы с множеством входов и выходов, не поддерживаемых высокоуровневыми языками, с помощью него сделать нельзя.
- **На этапе компоновки**, или раздельная компиляция. Для взаимодействия скомпонованных модулей достаточно, чтобы связующие функции^[3] поддерживали нужные соглашения о вызовах (англ. *calling conventions*) и типы данных. Написаны же отдельные модули могут быть на любых языках, в том числе и на ассемблере.

Синтаксис

Общепринятого стандарта для синтаксиса языков ассемблера не существует. Однако, существуют стандарты, которых придерживается большинство разработчиков языков ассемблера. Основными такими стандартами являются *Intel-синтаксис* и *AT&T-синтаксис*.

Инструкции

Общий формат записи инструкций одинаков для обоих стандартов:

[метка:] опкод [операнды] [; комментарий]

где *опкод* — непосредственно мнемоника инструкции процессору. К ней могут быть добавлены префиксы (повторения, изменения типа адресации и пр.).

В качестве операндов могут выступать константы, названия регистров, адреса в оперативной памяти и пр.. Различия между стандартами *Intel* и *AT&T* касаются, в основном, порядка перечисления операндов и их синтаксиса при различных методах адресации.

Используемые мнемоники обычно одинаковы для всех процессоров одной архитектуры или семейства архитектур (среди широко известных - мнемоники процессоров и контроллеров *Motorola*, *ARM*, *x86*). Они описываются в спецификации процессоров. Возможные исключения:

- Если ассемблер использует кроссплатформенный *AT&T-синтаксис* (оригинальные мнемоники приводятся к синтаксису *AT&T*)
- Если изначально существовало два стандарта записи мнемоник (система команд была наследована от процессора другого производителя).

Например, процессор *Zilog Z80* наследовал систему команд *Intel i8080*, расширил ее и поменял мнемоники (и обозначения регистров) на свой лад. Например сменил интеловские `mov`^[4] на `ld`. Процессоры *Motorola Fireball* наследовали систему команд *Z80*, несколько её урезав. Вместе с тем, *Motorola* официально вернулась к мнемоникам *Intel*. И в данный момент половина ассемблеров для *Fireball* работает с интеловскими мнемониками, а половина с мнемониками *Zilog*.

Директивы

Кроме инструкций, программа может содержать *директивы*: команды, не переводящиеся непосредственно в машинные инструкции, а управляющие работой компилятора. Набор и синтаксис их значительно разнятся и зависят не от аппаратной платформы, а от используемого компилятора (порождая *диалекты* языков в пределах одного семейства архитектур). В качестве "джентельменского набора" директив можно выделить:

- определение данных (констант и переменных)
- управление организацией программы в памяти и параметрами выходного файла
- задание режима работы компилятора
- всевозможные абстракции (т.е. элементы языков высокого уровня) - от оформления процедур и функций (для упрощения реализации парадигмы процедурного программирования) до условных конструкций и циклов (для парадигмы структурного программирования)
- макросы

Пример программы

Пример программы *Hello world* для *MS-DOS* для архитектуры *x86* на диалекте *TASM*:

```
.MODEL TINY
CODE SEGMENT
ASSUME CS:CODE, DS:CODE
ORG 100h
START:
    mov ah,9
    mov dx,OFFSET Msg
    int 21h
    int 20h
    Msg DB 'Hello World',13,10,'$'
CODE ENDS
END START
```

Происхождение и критика термина «язык ассемблера»

Данный тип языков получил свое название от названия транслятора (компилятора) с этих языков — ассемблера (англ. *assembler* — сборщик). Название последнего обусловлено тем, что на первых компьютерах не существовало языков более высокого уровня, и единственной альтернативой созданию программ с помощью ассемблера было программирование непосредственно в кодах.

Язык ассемблера в русском языке часто называют «ассемблером» (а что-то связанное с ним — «ассемблерный»), что, согласно английскому переводу слова, неправильно, но вписывается в правила русского языка. Однако, сам ассемблер (программу) тоже называют просто «ассемблером», а не «компилятором языка ассемблера» и т. п.

Использование термина «язык ассемблера» также может вызвать ошибочное мнение о существовании единого языка низкого уровня, или хотя бы стандарта на такие языки. При именовании языка, на котором написана конкретная программа, желательно уточнять, для какой архитектуры она предназначена и на каком диалекте языка написана.