***Ассемблер –*** компьютерная программа, компилятор исходного текста программы, написанный на языке ассемблера, в программу на машинном языке

***Ассемблер –*** программа, преобразующая исходные коды языка ассемблера в машинные команды

***Характеристики***

* Программная модель абстрактного ассемблера
* Организация оперативной памяти
* Основные типы данных
* Основные форматы команды
* Основные регистры, достаточные программисту
* Способы адресации
* Системы команды

***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 языке ассемблера**)

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

***Язык машинных кодов -***  запись машинной программы в виде последовательности восьмеричных или шестнадцатеричных цифр, где группа цифра задает значение байта или слова машинной программы

***Машинный язык (ЯМК) –*** управляющий код для конкретной машинной операции (команды), определяющий, откуда взять исходные данные и куда поместить результаты выполнения операции

***Пример машинной команды:***

1011 0000 0000 0101

Процессор Intel помещает 5 в регистр AL: первые 8 битов (1011 0000) – код операции

(предписываем процессору поместить 8-разрядное число в регистр AL)

Вторые 8 битов (0000 0101) это операнд (число 5), который необходимо поместить в регистр AL

***Язык программирования низкого уровня –*** это язык программирования, структура команд которого определяется форматом команд и данных машинного языка, а также архитектурной ЭВМ

Ярким представителем языка программирования низкого уровня является язык Ассемблера, который был разработан в 50-е годы прошлого века и позволяет писать программы с использованием специальных обозначений машинных кодов – мнемоники. Ассемблер широко применяется в программах, где необходимо высокое быстродействие

Ассемблеры ориентированы на определенные типы процессоров. Поскольку разные тип ЭВМ имели разные системы команд процессора, то и ассемблеры у них были разные. Поэтому Ассемблеры и является *машинно-ориентированным языком*. Такие программы нельзя переносить для исполнения на другие типы ЭВМ.

***Язык ассемблера -***  машинно-ориентированный язык записи программы в виде последовательности операторов, каждый из которых представляет либо директиву ассемблера, либо машинную команду

Обладая достаточно ограниченным набором команд и действий, язык ассемблера является универсальным средством, если вы хотите создать максимально быстро исполняемый код, а также пошагово отследить процессы, выполняемые машиной

Исходя из этого выделим очевидные **ДОСТОИНСТВА**:

***Быстродействие.*** Быстрее только использовать непосредственные инструкции процессора

***Безопасность.*** Низкоуровневость в данном случае практически исключает наличие белых пятен в коде

***Эффективность*** использования возможностей конкретной платформы. Ориентированность на используемую машину позволяет иметь серьезное преимущество по сравнению с высокоуровневыми языками

***Понимание исполняемого кода программистом.*** На таком уровне программирования код и действие имеют очевидную причинно-следственную связь

Выделим очевидные **НЕДОСТАТКИ**:

1. ***Трудно выучить.*** Специализация на платформу, несколько диалектов – все это не способствует быстрому изучению и пониманию
2. ***Тяжело читать.*** Большой листинг, простые однотипные операции
3. ***Написание программ.*** Некоторые примитивные операции, описываемые в языках высокого уровня одной строкой, здесь могут вызывать настоящую головную боль у программиста
4. Большинство используемых программ машин просто не нуждается в таком примитивном языке, как ассемблер
5. Высокий порог входа. Представить исполняемый код в языке ассемблера сможет почти любой программист средней руки. Сделать это эффективнее компилятора – лишь малая часть
6. ***Ограниченное*** количество библиотек, сообществ, вспомогательных ресурсов по современным меркам

***Синтаксис программы на языке ассемблера определяется*** главным образом системой команд конкретного процессора и системой директив конкретного транслятора

Для некоторых платформ может существовать несколько видов синтаксиса языка ассемблера, не совместимых между собой

Например, наиболее популярные синтаксисы языком ассемблера для Intel-совместимых процессоров – Intel-синтаксис и AT&T-синтаксис

***Виды предложений языка Ассемблера***

***Инструкции или команды –*** представляющие собой символические аналоги машинных команд. В процессе трансляции инструкции ассемблера преобразуются в соответствующие команды систем команд макропроцессора

***Макрокоманды –*** оформляемые определенным образом предложения текста программы, замещаемые во время трансляции другими предложениями

***Директивы –*** указание транслятору ассемблера на выполнение некоторых действий. У директив нет аналогов в машинном представлении

Различие между командами и директивами проявляется при ассемблировании

Директива – утверждение, которое выполняется ассемблеры во время трансляции исходной программы и получение машинных кодов

Команды Ассемблера всегда генерируют машинный код

***Комментарии***

**Формат команд**

|  |  |  |  |
| --- | --- | --- | --- |
| Метка | Мнемокод | Операнды | ; Комментарии |

***Имя метки –*** идентификатор, значением которого является адрес первого байте того предложения исходного текста программы, которое он обозначает. Имя интерпретируется как метка, переменная или ключевое слово, поэтому за ним всегда следует символ двоеточий «:»

***Мнемокод –*** мнемоническая запись кода операции машинной команды, используемая в языке ассемблера

***Операнд –*** данное, используемое при выполнении команды. Наиболее важными параметрами операнда являются длина (обычно в байтах) и местоположение (регистр процессора, ячейка памяти, порт ввода-вывода)

***Комментарии*** строки комментариев, содержащие любые символы, в том числе и буквы русского алфавита. Комментарии игнорируются транслятором

**Формат директивы**

***Имя –*** идентификатор, отличающий данную директивы от других одноименных директив

***Мнемокод -*** мнемоническая запись кода операции машинной команды, используемая в языке ассемблера

***Операнд –*** части команды, макрокоманды или директивы ассемблера, обозначающие объекты, над которыми производятся действия. Операнды ассемблера описываются выражениями с числовыми и текстовыми константами, метками и идентификаторами переменных…

Программы на языке Ассемблер может начинаться с НЕОБЯЗАТЕЛЬНОГО элемента – заголовка. Директива **Title**

***Упрощенный формат модели сегментации памяти***

**.MODEL** [разрядность] **модель** [язык]

Директива **.stack** является директивой организации структуры памяти и **устанавливает размер пространства для стека**

.stack 100h

Директива **.data** – является директивной организации структуры памяти и начинает или продолжает сегмент данных, где хранятся переменные

Директива **.code** – является директивой организации структуры памяти и отмечает начало сегмента кода, где должны находиться выполняемые команды

Директива **proc** – объявляет начала процедуры

endp – конец процедуры

Обращение к данным в программе

**mov ax, @data**

**mov ds, ax**

@data – имя, данное сегменту данных

Эти строки являются обязательными для любой программы

Компьютер – универсальный исполнитель, который умеет управлять другими исполнителями и обладает собственной внутренней памятью

Запись алгоритма для компьютера называется программой

Среди устройств и узлов, входящих в состав компьютера, наиболее важными для выполнения любой программы являются:

* Центральный микропроцессор (который для краткости в дальнейшем будем называть просто процессор)
* Оперативная память – оперативное запоминающее устройство ОЗУ
* Внешнее запоминающее устройство (ВЗУ)

В оперативной памяти хранится выполняемая программа вместе с принадлежащим ей данными

Операционная система, загрузив программу и при необходимости настраивает ее для выполнения в той области памяти, куда она попала, сообщает процессору начальный адрес загруженной программы и инициирует процесс ее выполнения

Процессор выполняет вычисления и другие действия, описанные в программе

Все современные компьютеры построены с использованием концепции, разработанной американским ученым венгерского происхождения Джоном фон Нейманом, содержащий следующие принципы:

Компьютер должен состоять из следующих модулей: управляющий блок (контроллер), арифметический блок, память, блоки ввода\вывода

* Строение компьютера не должно зависеть от решаемой задачи, программа должна храниться в памяти
* Инструкции и их операнды (то есть данные) должны также храниться в той же памяти
* Память делится на ячейки одинакового размера, порядковый номер ячейки считается ее адресом (1 ячейка эквивалентна 1 байту)
* Программа состоит из серии элементарных инструкций, которые обычно не содержат назначения операнда (указывается только его адрес), поэтому программу не зависит от обрабатываемых данных (это уже прототип переменных). Инструкции выполняются одна за другой, в том порядке, в котором они находятся в памяти
* Для изменения порядка выполнения инструкций используются инструкции условного или безусловного (jump) перехода
* Инструкции и данные (то есть операнды, результаты или адреса) представляются в виде двоичных сигналов и в двоичной система счисления

Процессор – кремниевая плата или «подложка» с логическими цепями, состоящими из транзисторов, скрытая в пластмассовом корпусе, снабженном контактными ножками (выводами pin). Большинство ножек процессора подключено к шинам – шине адреса, шине данных и т.д.

Арифметико-логическое устройства выполняет арифметические и логические действия над данными.

Центральный процессорный блок последовательно выполняет три основных команды: выборку, декодирование, выполнение.

Выборка – процессор считывает из памяти первую команду программу. При этом он: помещает очередную команду в участок, называемый очередью, обновляет программный счетчик

Декодирование – находит в памяти данные, необходимые для ее выполнения (если команда требует данных). При этом он: рассчитывает адрес, выбирает операнды из памяти

Выполнение – выполняет необходимые расчеты, сохраняет результаты в памяти или регистрах, устанавливает необходимое состояние флагов

Регистры представляют собой сверхоперативную память небольшого размера, которая предназначен для временного хранения служебной информации или данных

Регистра подразделяются на: регистры общего назначения, сегментные регистры, регистры состояние, индексные регистры

Регистр AX – аккумулятор, используется при хранении промежуточных данных.

Регистр BX – базовый регистр, единственный регистр общего назначения, используемый в индексной адресации, применяется для хранения базового адреса некоторого объекта в памяти, содержит адреса процедур или переменных, используется при вычислениях

Регистр CX – счетчик для операций повторения и циклических команд. При выполнении этих команд увеличение значения происходит автоматически. Используется при вычислениях.

Регистр DX – регистр данных, используется в некоторых операциях ввода\вывода, в операциях умножения и деления 16-ти разрядных чисел совместно с регистром AX, в него помещается результат операции

Обзор системы команд процессора:

Команды передачи данных – команды этой группы используются для перемещения данных между различными узлами компьютера

Арифметические команды – предназначены для выполнения сложений, вычитаний, умножений и делений чисел

Логические команды – выполняют логические сложение и умножение, инверсию и т.д.

Команды сдвигов – выполняют различные разрядные сдвиги влево и вправо

Команды управления выполнением – в эту группу включаются команды, нарушающие естественный порядок выполнения команд программы, такие как программные прерывания и возврат из прерываний, вызов и возврат из подпрограмм, различные условные и безусловные переходы

Операнд – объект, над которым выполняется машинная команды.

mov ax, bx; bx – операнд источник; ax – операнд приемник

Основные типы операндов:

* Непосредственный – константа
* Регистр – значение одного из регистров процессора
* Память – ссылается на значение в памяти

Содержимое регистра eflags – флаги состояния

Эти флаги могут изменяться после выполнения машинных команд

Флаги состояния регистра eflags отражают особенности результата исполнения арифметических или логических операций

Это дает возможность анализировать состояние вычислительного процесса и реагировать на него с помощью команд условных переходов и вызовов подпрограмм

CF – флаг переноса

SF – флаг знака

0F – флаг переполнения

ZF – флаг нуля

AF – флаг дополнительного переноса

PF – флаг четности

Системные флаги

Команда int

Регистр указателя команд IP, указывает в каждый момент относительный адрес команды, следующей за исполняемой, программно недоступен

CS – регистр сегмента кода

(<CS:IP>)

Команды передачи управления – нарушить заданный порядок, обойти некоторый участок программы, перейти на выполнение другой ветви, передать управление подпрограммы

Команды передачи управление: безусловные (без условия) и условные (зависит от условий)

Безусловные: собственные (без возврата в точку перехода), вызовы подпрограмм (с возвратом после завершения подпрограммы), вызовы программных прерываний и возврат из программных прерываний

JMP метка

<IP> = <IP> + смещение к нужной команде

JMP Back, JMP Forward

short – прямой короткий переход

near ptr – прямой ближний переход

word ptr – косвенный ближний переход

jmp short shpt – переход на метку shpt в пределах -128 – 127

jmp shpt – то же самое

jmp near ptr pt – переход на метку pt в пределах текущего сегмента

jmp pt – то же самое

Цикл на основе JMP:

Start: Mov ah, 2

Mov dl, ‘A’

Int 21h

JMP Start

CMP получатель, отправитель – сравнение, флаги CF (1, если получатель < отправителя) и ZF (>)

Jcond метка – команда перехода по условию передает управление по указанному адресу, когда признак условия установлен

E – равно

N – не равно

G - больше

L - меньше

A – выше по адресу

B – ниже по адресу

Простое утверждение if:

cmp op1, op2

JE True

Jmp End\_IF

True:

<утверждение1>

<утверждение n>

END\_IF:

Использование оператора OR:

CMP AL, op1

Jg L1

CMP AL, op2

Jge L1

CMP AL, op3

Je L1

CMP AL, op4

Jmp L2

L1: <утверждение>

L2:

Использование оператора AND:

CMP AL, op1

Jng L1

CMP Al, op2

Jnge L1

CMP AL, op3

Jne L1

CMP Al, op4

Jnl L1

<утверждение>

L1:

Структура CASE:

Mov al, input

Cmp al, ‘a’

Je proc-A

Cmp al, ‘B’

Je proc\_B

WHILE:

Do\_while:

Cmp op1, op2

Jnl end\_do

<com1>

<com2>

Jmp do\_while

End\_do:

OR AX, AX – проверка на знак

NEG AX – изменение знака

SUB DX, DX – обнуление регистра

ADD DX, ‘0’ – перевод в символ цифры

MOV CX, N

LOOP:

DEC CX

CMP CX, 0

JNE LOOP

LOOP метка\_перехода

Вычитает 1 из CX

Сравнивает CX с 0

Если CX = 0, то управление передается на следующую метку

Intarray dw 0, 0, 0, 0, 1, 20, 35, -12, 66, 4, 0

Array\_Size = ($ - Intarrat) / 2