mount с d:\prax

где "d:\prax " – это папка на реальном диске D, в которой должны быть свалены воедино ваши com-примеры и файлы отладчика CV. А "c" – это уже виртуальный диск внутри dosbox'a.

Далее переходим на новый виртуальный диск командой

с:

FC [/A] [/C] [/L] [/LBn] [/N] [/OFF[LINE]] [/T] [/U] [/W]

[/nnnn][диск1:][путь1]имя\_файла1 [диск2:][путь2]имя\_файла2

FC /B [диск1:][путь1]имя\_файла1 [диск2:][путь2]имя\_файла2

В описании командных параметров принято использовать разные скобки, и вот что они означают:

<обязательно с предыдущем элементом>

[не обязательно]

Обязательно {либо это | либо это}

Команда CD. Cправку о ней можно получить, набрав её с ключом вопроса без пробела:

cd/?

Они позволяют дополнить поведение команды.

Выше я уже использовал оператор переназначения вывода ">".

Сейчас покажу ещё пару полезных операторов:

nul - ничего

\* - всё что угодно

Допустим, нам нужно скопировать все файлы из папки Windows\system32, которые начинаются на ms (а дальше может быть любое сочетание символов в именах и любое расширение) на диск D в папку MS. Вот как это делается с помощью команды COPY:

copy C:\Windows\system32\ms\*.\* D:\MS

Очень удобно с оператором \* обрабатывать все файлы с одним расширением (например \*.txt) или искать сочетание букв (например \*ms.\*).

Кстати, в FAR'e удобно выделять желаемое, используя клавишу "+" на нампаде.

Оператор nul действительно означает "ничего", допустим, строка:

tree >nul

приведёт к тому, что вывод команды tree уйдёт в никуда.

Так можно указывать ещё и nul-файлы (мнимые) в командной строке.

Для чего вам нужно "ничего" - соображайте сами =).

Рассмотрим пару простых bat-примеров.

@echo off

echo Список действующих прописных путей:

echo %PATH%

Интересно то, что исполняемый файл (com или даже exe) можно писать в простом текстовом редакторе, вот сейчас и попробуем.

Эксперимент 00 (prax00.com)

Создайте файл (в FAR'e - Shift+F4). Дайте этому файлу любое имя с расширением "com" (простой тип исполняемых файлов). Допустим, файл будет называться prax00.com.

В этом файле можно напечатать несколько латинских ПРОПИСНЫХ букв "A" или русских ПРОПИСНЫХ букв "P". Или и то и другое, можно вперемешку.

Только русские "Р" должны быть в кодировке DOS. (в редакторе FAR'а нажмите несколько раз F8, когда сверху увидите 866 - это и означает, что включена кодировка IBM ASCII).

Всё должно быть без пробелов и Enter'ов - только английская буква "ЭЙ" и русская "ЭР". Ну, можете ещё для разнообразия накидать других латинских букв. :)

Сохраните файл (в FAR'e - F2).

! Не запускайте пока этот пример.

Единственное, что отличает prax00.com в данный момент от настоящей программы, - это отсутствие команды завершения программы. Её-то мы сейчас и добавим.

Нужно ввести два "волшебных" символа - двойную рамку и обычный пробел.

Как это сделать? Есть несколько способов:

Способ I. Можно просто ввести код символа с клавиатуры (Gambit, спасибо за идею). Чтобы получить символ двойной рамки, нажмите Alt, на калькуляторном блоке клавиатуры наберите "205" и затем отпустите Alt, должен появится символ похожий на "=".

Способ II. По F8 в Far'e переключитесь в кодировку 1251 и добавьте ПРОПИСНУЮ русскую "Н" (она в кодировке Windows1251 имеет тот же код 205).

Способ III (самый важный). Закройте редактор Far'a и откройте этот файл в Hiew'e (в командной строке наберите "Hiew32 prax00.com") или просто выберите файл в меню Hiew'a). Сейчас вы видите обычные буквы, но если нажмёте "F4" (Mode), то увидите байты в hex-виде. Примерно так это должно выглядеть:

Адреса Маш.команды Команды Асма

Байты Имена Операнды

00000000: 90 nop

00000001: 41 inc cx

00000002: 90 nop

00000003: 41 inc cx

00000004: 90 nop

00000005: 90 nop

00000006: 41 inc cx

00000007: 41 inc cx

00000008: 42 inc dx

00000009: 42 inc dx

0000000A: 42 inc dx

0000000B: CD20 int 020

Адрес - это номер байта в файле, который занимает начало инструкции (примечание). В этом примере все маш.команды занимают 1 байт, кроме последней.

Байты машинных команд - машинный код в шестнадцатиричном виде. Одна строка - одна команда процессора.

Имена - мнемоники команд (символические имена команд Ассемблера).

Операнды - объекты, над которыми производится действие.

Позже мы всё разберём по винтикам.

Сейчас здесь есть только одна "полезная" команда "int 20".

90hex (или 90h) - это цифровой код операции (опкод) команды NOP. Так что же делает "nop", спросите вы.

НИЧЕГО!

NOP - (No oPeration - нет операции)

Суть этой команды в ничегонеделаньи. Это очень полезная команда, ею можно пользоваться в разных ситуациях, вы сами потом поймёте, что она нужна.

А что такое inc CX?

Inc - это команда, которая увеличивает значение на 1 (INCrement - инкремент)

Байт это 8 бит, и что самое главное, байт - минимально адресуемая ячейка памяти.

Если нужно прочитать информацию, например, из бита 900, то нам нужно обратиться к 112-му байту и посмотреть в нём бит номер 4.

| Адрес в байтах | Информация в БИТАХ

| | 76543210 - номера бит (разряд)

------|-------------------|-----------------------------------------------

111d | 0000006F | 00000000

112d | 00000070 | 000?0000

113d | 00000071 | 00000000

114d | 00000072 | 00000000

Конечно же, в компьютере физически биты не разделяются пробелами. Вся оперативная память, например, - сплошной поток выключателей :).

Но при отображении биты обычно разделяют на:

байты - 8 бит, две hex-цифры, или

тетрады - 4 бита, одна hex-цифра.

Предмет, с (или над) которым производится действие, называется операнд.

Операнды в ассемблере для Интел-совместимых процессоров принято разделять запятыми. То есть в некоторых системах или в других языках программирования пишут:

AX xor 44

или вполне может быть такая форма записи:

44,55 xоr AX

Но в x86 ассемблере принято писать так:

xor AX,44

где AX - операнд 1 (он же приёмник),

а 44 - операнд 2 (он же источник).

В Hiew'e (когда вы видите дизассемблерный код нашего файла) нажмите F3 и затем Enter. Теперь можно набирать программу на ассемблере (символ "1" в файле должен стереться). Каждая инструкция вводится Enter'ом и превращается в строку, если нет явной ошибки. Пробелы нужны только для удобства, поэтому неважно, сколько их. Пишите как хотите, строчными или прописными буквами, но только по-англицки. :)

Вот код программы, его нужно набрать:

mov ah,9

mov dx,10Dh

int 21h

mov ah,10h

int 16h

int 20h

Когда всё напишете, нажмите один раз Esc, чтобы прекратить ассемблирование, и F9, чтобы сохранить файл.

Это был весь код программы, которая должна выводить строку на экран! Круто, правда? Только не хватает самой строки.

Для того, чтоб вписать строку, нужно открыть файл в текстовом редакторе (в FAR'e - F4).

Допишите после всех закорючек (только не сотрите ничего) любую текстовую строку и в конце поставьте знак $.

Это может выглядеть примерно так:

\_?\_?

\_?\_?\_?\_?\_Good Day!$

дреса Маш.команды Команды Асма комментарии

Байты Имена Операнды

00000000: B409 mov ah,009 ; Поместить значение 9 в регистр AH (параметр1)

00000002: BA0D01 mov dx,0010D ; Поместить адрес текстовой строки в DX (параметр2)

00000005: CD21 int 021 ; Вызвать подпрограмму, в которой

; отработает функция вывода текста на экран (AH=09)

00000007: B410 mov ah,010 ; Поместить значение 10h в регистр AH (параметр1)

00000009: CD16 int 016 ; Вызвать подпрограмму ожидания нажатия клавиши

0000000B: CD20 int 020 ; Подпрограмма завершения

0000000D: 47 inc di

0000000E: 6F outsw

0000000F: 6F outsw

00000010: 64204461 and fs:[si][61],al

00000014: 7921 jns 000000037 ---X

00000016: 24 and al,000

Итак, почему же ah?

AH - это регистр процессора.

Регистры - это несколько байт, которые физически находятся в центральном процессоре (ЦП). Регистры нужны для вычислений и связи ЦП с внешним миром.

Большая часть кода программ состоит из команд копирования значений из оперативной памяти в регистры и обратно.

Знаю по себе: понять, зачем нужны регистры можно только через практику. Другого способа нет.

Причём где-то дней 10-15 я вообще не врубался - как оно всё... чтобы вот как-то вот так. :)

Но сейчас мне кажется, что регистры - самое простое, и в то же время важное, что есть в процессоре.

Ну, сами подумайте, что делает программа?

Приём данных от пользователя (данные передаются в регистры).

Вычисления (загрузили в регистр число и дали команду, как его обработать).

Приём данных из устройств (опять они - регистры).

Изменение состояния ЦП и периферийных устройств (только регистры! Например, регистр флагов).

Вывод данных на устройства.

Передача данных другим программам.

И т.д.

В окне 3 - дизассемблированный код.

В окне 5 - hex-байты. Их адреса (слева) и символы, которые они означают (справа).

В окне 7 - регистры процессора.

9 - командная строка.

Главная кнопка здесь будет F10, вам нужно нажимать только на неё (всё остальное можно делать мышкой). Эта кнопка с каждым нажатием будет выполнять текущую строку.

Вы уже понимаете, что отображается в окне[5], и также имеете представление, что в окне[3] - код программ. А вот окно [7] для вас пока ничего не означает, и это очень досадно, но мы сейчас исправимся.

Нажмите F10 один раз. Выполнится строка

mov ah,09

Эта команда означает: поместить значение 09 в регистр AH. Но в окне[7] нет регистра AH, а после выполнения этой строки изменится регистр EAX.

Было EAX=00000000

Стало EAX=00000900

Дело в том, что регистр AH - это часть регистра AX, а он - часть регистра EAX.

Если, предположим, мы загрузили регистр EAX значением 44332211, выглядеть он будет так:

EAX=44332211

AX= 2211

AH= 22

AL= 11

Один 32-разрядный регистр - это всего лишь 4 байта в процессоре.

Точно так же устроены ещё 3 регистра.

Схематично их отображают так:

EAX EBX ECX EDX

AX BX CX DX

E-часть AH AL E-часть BH BL E-часть CH CL E-часть DH DL

EAX (сокращение от Accumulator)

EBX (сокращение от Base)

ECX (сокращение от Counter)

EDX (сокращение от Data)

Можно, кстати, писать и строчными буквами, регистры не обидятся.

EAX, EBX, ECX и EDX - 32-битные регистры данных центрального процессора (от 386-го и по сей день). В эти регистры помещаются необходимые значения, и в них же чаще всего оказываются результаты вычислений. В 32 битах можно хранить hex-число от 00 00 00 00h до FF FF FF FFh. И это всё, что может там храниться. На назначения (Accumulator, Base, Counter, Data) в наших уроках можно вообще не обращать внимания. Когда вы освоите команды Ассемблера, вы сами поймете, почему Counter и почему Base.

Первая строка кода из нашей программы уже почти понятна.

mov ah,9

Имя команды операнд1,операнд2

где операнд1 - регистр AH

операнд2 - цифра 9

Остаётся непонятной только сама команда.

MOV - это основная команда Ассемблера, она встречается в программах гораздо чаще остальных.

Для программирования самое важное понятие - переменная. На самом деле переменная лишь абстракция для удобства программирования. Такая же абстракция, как в алгебре X, Y, Z. Чтоб решить какую-нибудь задачу, мы говорим, что Y будет равен 5, а Z будет равен 3.

Так вот присвоение переменной Y значения 5 на Ассемблере будет выглядеть так:

mov Y,5

Команда MOV

Происхождение от англ. слова move - движение, перемена места

Формат mov приёмник , источник

Действие Копирует содержимое источника в приёмник.

Примечание MOV не может передавать данные между двумя адресами оперативной памяти (для этой цели существуют команды MOVS)

Этого достаточно, чтоб использовать команду в своих программах. Посмотрите ещё раз на первые две строки prax01:

Имена Операнды комментарии

команд

mov ah,09 ; поместить значение 9 в регистр AH

mov dx,010D ; поместить значение 010Dh в DX

Поняли?

Получив первую инструкцию, процессор выполнит инициализацию своего 8-битного регистра AH значением 9, после чего регистр AH будет содержать только байт 09.

При выполнении второй команды процессор поместит в свой 16-битный регистр DX значение 010Dh, после чего регистр DX будет содержать только эти два байта 01 и 0Dh.

Причём, если вы ещё раз посмотрите на устройство регистров, вы обязательно поймёте следующее:

Так как регистр DX состоит из DH и DL, то можно сказать, что после выполнения второй строки кода программы в регистре DH окажется значение 01, а в регистре DL окажется значение 0Dh.

DH DL

DX=01 0D

Это просто, но важно! В 16-битный регистр (AX,BX,CX,DX) нельзя положить значение больше двух байт (FFFFh).

А в 8-битный (AH,AL, BH,BL, CH,CL, DH,DL) нельзя положить больше байта, то есть FFh.

И еще, допустим:

EAX=99884433

AX= 4433

AH= 44

AL= 33

Вы должны понять, что физически есть только 4 байта (99 88 44 33h). По отдельности можно обращаться к AX за значением 4433h, или к AH за 44h, или к AL за 33h. Но 9988h находится в E-части, а у неё нет собственного имени, она не является подрегистром. Вы не можете прочитать или загрузить 2 старших байта такого регистра, не обратившись ко всему регистру. Пример:

mov EAX, 0FFFFFFFFh ; Так правильно, и EAX будет равен FFFFFFFF

mov EAX, 01FFFFFFFFh ; Так НЕправильно. Значение больше регистра,

; данной операции быть не может

mov EAX, 0 ; Так правильно, и EAX станет равен 00000000

mov AX, 0FFFFh ; Так правильно, и EAX будет равен 0000FFFF

mov AX, 1FFFFh ; Так НЕправильно. Значение больше регистра,

; данной операции быть не может

mov AX, 0 ; Так правильно, и AX станет равен 0000

mov AH, 111h ; Так НЕправильно. Значение больше регистра,

; данной операции быть не может

mov AL, 100h ; Так НЕправильно. Значение больше регистра,

; данной операции быть не может

mov AL, 0BBh ; Так правильно, и EAX станет равен 000000BB

mov AH, AL ; так правильно, и EAX станет равен 0000BBBB

Регистры-указатели

Они тоже входят в группу РОН. В этой программе нам они безразличны, но далее мы научимся работать и с ними. Причём на практике всё будет очень похоже на регистры данных. Да и вообще отличий между ними совсем немного. Главное отличие в том, что в регистрах-указателях нет подрегистров, есть только одна вложенная часть.

ESP=44332211 (Extended Stack Pointer)

SP= 2211 (Stack Pointer)

Мы опять видим четыре байта, ведь все E-регистры 32-разрядные (32 бита - это 4 байта).

EBP включает в себя BP (Base Pointer) указатель базы

ESP включает в себя SP (Stack Pointer) указатель стека

ESI включает в себя SI (Source Index) индекс источника

EDI включает в себя DI (Deliver Index) индекс приёмника

Регистр адреса текущей машинной команды - EIP

Процессор берёт из памяти машинную команду и увеличивает текущий адрес так, чтобы он указывал на следующую команду. Именно для этого и существует EIP.

EIP=44332211 (Extended Instruction Pointer)

IP= 2211 (Instruction Pointer)

И опять посмотрите на первые две строки кода в отладчике:

Адрес Байты Имена Операнды

0100: B409 mov ah,009

0102: BA0D01 mov dx,0010D

Вот код нашей следующей программы.

Этот кусок исходника не для Hiew'a, прежде чем набирать программу, обязательно прочитайте пояснение ниже.

prax02.com:

mov BX, 0133h ; Поместить в BX значение 0133h (базовый адрес)

mov dword ptr [ВX],04030201h ; Поместить в память по адресу 0133h

; значение 01020304h размером в 4 байта

add dword ptr [BX],30303030h ; Прибавить к 4 байтам в памяти по адресу 0133h

; значение 30303030h, размером в 4 байта.

mov dx,bx ; Копировать содержимое

; регистра BX в регистр DX (BX=DX=133h)

mov ah,9 ; Поместить значение 9 в регистр AH

int 021h ; Вызвать подпрограмму вывода текста на экран

mov ah,10h ; Поместить значение 10h в регистр AH

int 16h ; Вызвать подпрограмму ожидания нажатия клавиши

mov dword ptr [BX],50505050h ; Поместить в память по адресу 0133h

; значение 50505050h размером в 4 байта

sub dword ptr [BX],18191A1Bh ; Отнять от 4 байт в памяти по адресу 0133h

; значение 1B1A1918h размером 4 байта

mov ah,9

int 021h ; Ещё один вывод строки

mov ah,10h

int 16h ; Повторить ожидание нажатия клавиши

int 020h ; Вызвать подпрограмму завершения

Хочу сказать ещё об одной важной особенности.

Все строки в чистом Ассемблере содержат только одну команду.

А что же тогда "dword ptr", спросите вы.

Dword ptr - не команда, это всего лишь определитель размера данных. Подобная запись, можно сказать, - надстроенная условность. После того как вы наберёте эту программу, мы обстоятельно поговорим про данные и их размер. А сейчас только скажу, что в Hiew'e вместо "dword ptr" пишется "d," (маленькая латинская буква "ди" и запятая без пробела) чтоб ещё больше сократить писанину.

В Hiew'e создать такую программу можно как и всегда. Создав отдельно com-файл, откройте его, переключитесь в режим Ассемблера (F4,F4 или Enter, Enter), начните редактировать (F3) и затем нажмите Enter, чтоб вводить команды.

Набрав последнюю инструкцию (int 020h), введите её Enter'ом и нажмите Esc, чтобы прекратить ассемблирование. Теперь впечатайте в самом конце 5 новых байтов 00 00 00 00 24 (ничего при этом не стирая выше). Сохраните файл (F9).

Когда программа загрузится в оперативную память, вместо нулей в ходе выполнения кода будут вписаны данные.

А вот байт 24h - это символ $. Так же, как и в первом примере, этот символ нужен для окончания строки текста.

Вот как в Hiew'e будет выглядеть эта программа.

Адреса Байты имена операнды комментарии

00000000: BB3301 mov bx,00133 ;" 3"

00000003: 66C70701020304 mov d,[bx],004030201 ;" "

0000000A: 66810730303030 add d,[bx],030303030 ;"0000"

00000011: 8BD3 mov dx,bx

00000013: B409 mov ah,009 ;" "

00000015: CD21 int 021

00000017: B410 mov ah,010 ;" "

00000019: CD16 int 016

0000001B: 66C70750505050 mov d,[bx],050505050 ;"PPPP"

00000022: 66812F1B1A1918 sub d,[bx],018191A1B ;" "

00000029: B409 mov ah,009 ;" "

0000002B: CD21 int 021

0000002D: B410 mov ah,010 ;" "

0000002F: CD16 int 016

00000031: CD20 int 020

00000033: 0000 add [bx][si],al

00000035: 0000 add [bx][si],al

00000037: 24 and al,000

00000003: 66C70701020304 mov d,[bx],004030201

Суть здесь в том, что когда операнд находится в квадратных скобках, это означает, что нужно производить действие по адресу в памяти, указанному операндом. Значит, эта строка просто записывает 4 байта (04 03 02 01h) в оперативную память по адресу 0133h. Но есть и запутка.

Самые любопытные сразу же спросят - а почему BX?

Так уж устроен машинный язык Intel.

BX не случайно, дело в том, что его назначение (база) как раз и состоит в адресации.

В реальном режиме, при указывании адреса мы здорово ограничены имеющимися инструкциями. И в данной ситуации мы можем использовать только SI, DI - регистры-указатели или BX - регистр базы (суть одна).

А вот в 32-битном режиме адресации такие инструкции есть для всех РОН, и мы можем использовать любой из них для подобных целей.

Всё это сложно и, самое главное, не очень важно. Вы просто можете попробовать набрать команду "mov d,[AX],004030201" и получить отказ. :)

После выполнения этой инструкции:

Память будет изменена

адрес значение

0133h 01

0134h 02

0135h 03

0136h 04

0137h 24h не меняется (24h - это ASCII-код символа "$")

0000000A: 66810730303030 add d,[bx],030303030

Здесь происходит сложение четырёх байт в памяти по адресу 0133h и четырёх байт, явно указанных в команде (30h30h30h30h). По адресу 0133h находится приёмник, а источник - непосредственно в команде.

После выполнения этой строки:

Память будет изменена

адрес старое новое

значение значение

0133h 01 +30h= 31h (31h - это ASCII-код символа "1")

0134h 02 +30h= 32h (32h - это ASCII-код символа "2")

0135h 03 +30h= 33h (33h - это ASCII-код символа "3")

0136h 04 +30h= 34h (34h - это ASCII-код символа "4")

0137h 24h не меняется (24h - это ASCII-код символа "$")

Сейчас можете не напрягать голову, так как команду int в Win32 мы пока использовать не будем.

Int - это не просто команда.

Здесь происходит вызов самостоятельной программы, такие программы называются прерываниями (interrupt). Давным-давно договорились, что для основных функций компьютера нужно создать набор стандартных программ, которые можно будет вызывать из любой другой программы. Так как при вызове таких внешних программ выполнение нашей приостанавливается, их обозвали прерываниями.

Ещё до загрузки какой-либо операционной системы можно использовать некоторые из них (прерывания BIOS). Это вовсе не означает, что они жестко заданы.

Существует специальная таблица, в которой хранятся векторы (адреса) прерываний. Можно самостоятельно задать вектор любого прерывания с номерами от 0 до FFh (больше не придумали). Так и делает любая ОС.

Во время загрузки DOS происходит переназначение (инициализация) таблицы векторов прерываний.

Так уж условились, что прерывание 21h будет занято DOS-функциями. Номер функции сообщается через AH. Там сейчас значение 09 (вывод строки на экран). Строка для вывода будет взята по адресу, сообщённому в DX (опять же чистая условность в ДОС). На самом деле у int 21h очень много функций. Прерывание 21h так и называют - мультифункциональное DOS-прерывание.

В Windows-программах практически не используются прерывания (только в исключениях). В форточках - свои функции, называемые API (об этом дальше).

Когда процессор получает команду "int номер", он переходит к выполнению вызванного прерывания.

Что же происходит внутри подпрограммы?

Прерывание 21h проверяет значение регистра AX, и если AH =9, выполняется функция вывода строки на экран. Её код делает довольно простое действие - копирование в видеопамять участка памяти, начиная от адреса из DX и заканчивая знаком "$". Сделав это, подпрограмма передаст управление обратно нашей программе со следующей строки.

После выполнения этой строки:

На экране в окне DOS должна появиться текстовая строка

1234