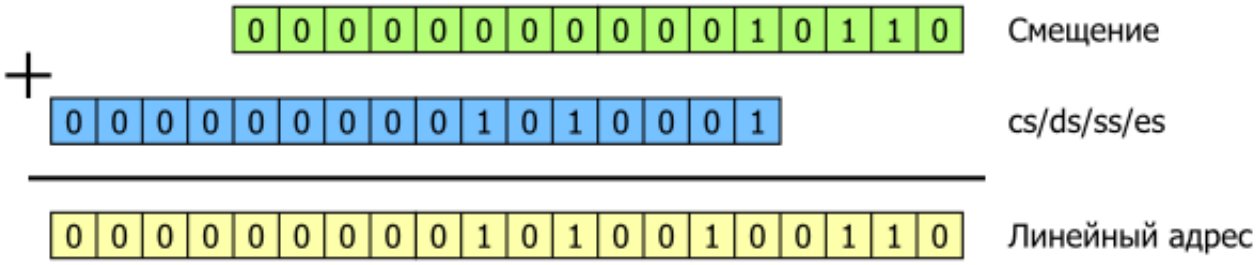
Учебный курс. Часть 31. Сегментная адресация

Автор: xrnd | Рубрика: Учебный курс | 14-04-2011 | Распечатать запись

До этой части учебного курса мы создавали только COM-программы, в которых один и тот же сегмент использовался для кода, данных и стека. Однако, для дальнейшего изложения необходимо подробнее разобраться с сегментной адресацией.

Формирование адреса в реальном режиме

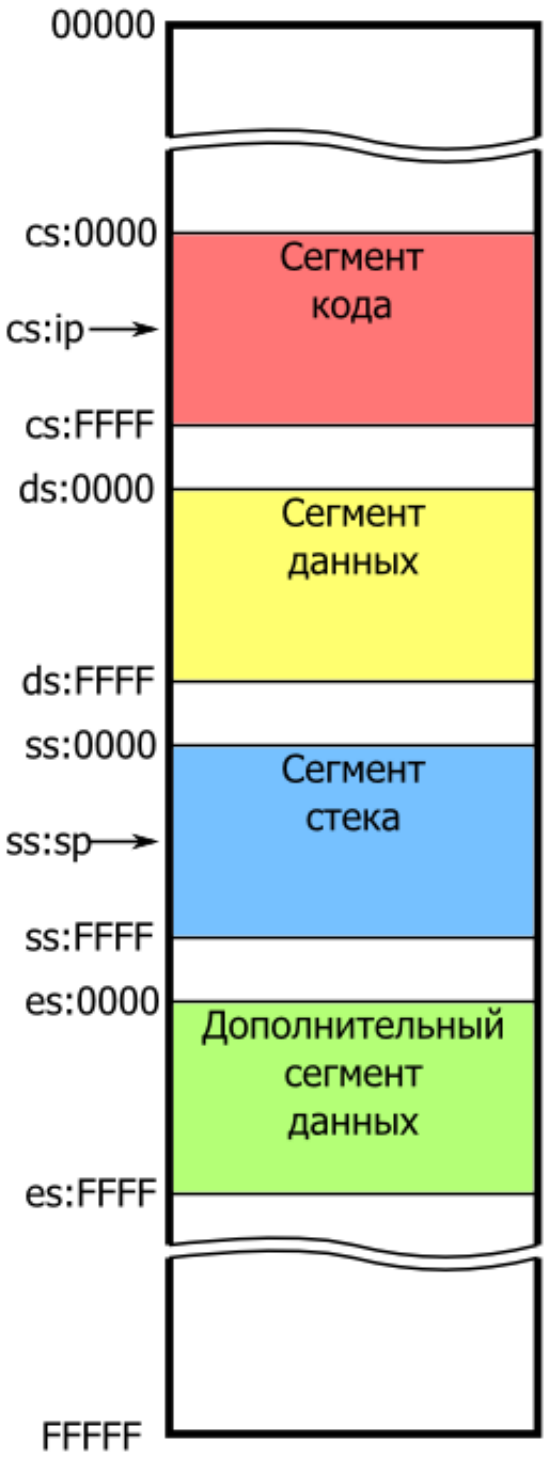
Основная идея сегментной адресации в том, что адрес состоит из двух частей — сегментной и смещения. Обычно их записывают через двоеточие (например 0100:0500). Линейный адрес любой ячейки памяти получается в результате сложения смещения и сегментной части, сдвинутой на 4 бита влево.



Начало сегмента всегда выровнено на границу параграфа (адрес кратен 16 байтам). Максимальный размер сегмента равен 2^{16} = 64 КБайта. А всего можно адресовать 2^{20} = 1 МБайт памяти. Конечно, сейчас такой объем памяти кажется смешным, но раньше это было очень много 🙂

Одна из особенностей сегментной адресации — неоднозначность представления адреса. Допустим, требуется обратиться к ячейке памяти по адресу 00400. Этот адрес может быть представлен как 0000:0400, 0040:0000, 0020:0200 и так далее.

Загруженная в память программа может одновременно работать с четырьмя сегментами. Сегменты могут перекрываться или даже совпадать, как это было в случае с COM-программой.



Для всех команд подразумевается сегмент «по умолчанию». Например, команды PUSH и POP работают с сегментом стека. Если операнд такой команды находится в памяти, то он берётся из сегмента данных. Команды JMP и LOOP вычисляют адрес перехода в сегменте кода.

Создание DOS EXE

Возможности сегментной адресации полностью реализуются в исполняемом файле DOS EXE. Не путайте этот формат с исполняемым файлом Windows (PE EXE)! Расширение такое же, но файл имеет совершенно другую структуру.

```
1 format MZ                     ;Исполняемый файл DOS EXE (MZ EXE)
2 entry code_seg:start          ;Точка входа
3 stack 200h                    ;Размер стека
4
5 .
```

```
;-----------------------------------------------------------------------
segment data_seg                ;Сегмент данных
hello db 'Hello, asmworld!$'    ;Строка

;-----------------------------------------------------------------------
segment code_seg                ;Сегмент кода
start:                          ;Отсюда начинается выполнение программы
    mov ax,data_seg             ;\
    mov ds,ax                   ;/ Инициализация регистра DS

    mov ah,09h                  ;\
    mov dx,hello                ; > Вывод строки
    int 21h                     ;/

    mov ax,4C00h                ;\
    int 21h                     ;/ Завершение программы
```

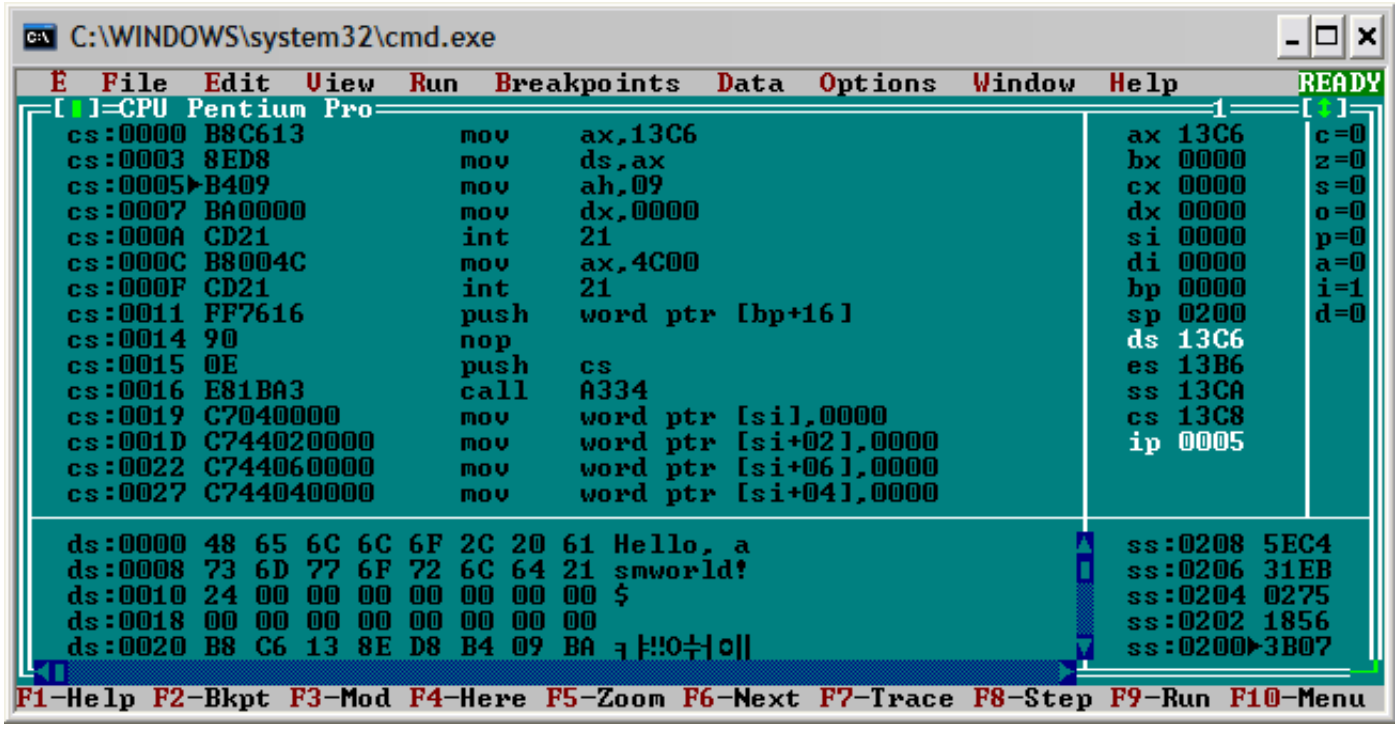
В первой строке после директивы *format* нужно поставить *MZ*, чтобы FASM сгенерировал нужный нам файл.

Во второй строке указывается точка входа, то есть метка, с которой начинается выполнение программы. Имя метки дополняется названием сегмента, в котором она находится.

После директивы *stack* можно указать требуемый размер сегмента стека в байтах (по умолчанию используется 4096). Дальше файл состоит из сегментов, которые объявляются с помощью директивы *segment*. После директивы записывается название сегмента.

В моём примере файл состоит из двух сегментов. В первом находятся данные (а точнее строка), во втором — код. Выполнение программы начинается с метки *start* в сегменте кода. После запуска необходимо инициализировать регистр *ds*, чтобы выбрать нужный сегмент данных. Для этого используются 2 команды, так как невозможно напрямую записать значение в сегментный регистр (нет команды MOV *ds*,*значение*).

Работу программы с сегментами можно увидеть в отладчике. Обратите внимание, что *cs*, *ds*, *es* и *ss* имеют разные значения:



Префиксы переопределения сегментов

Иногда нужно изменить используемый сегмент данных только для одной команды. Например, хочется прочитать байт из текущего сегмента кода или записать в сегмент стека. Для этого предназначены префиксы переопределения сегмента. Названия префиксов совпадают с названиями сегментных регистров.

Особенность синтаксиса FASM в том, что префикс пишется внутри квадратных скобок (так как по смыслу является частью адреса):

```
format MZ                       ;Исполняемый файл DOS EXE (MZ EXE)
entry code_seg:start            ;Точка входа
stack 200h                      ;Размер стека

;---------------------------------------------------------------------
segment data_seg                ;Сегмент данных
hello db 'Hello, asmworld!$'    ;Строка

;---------------------------------------------------------------------
segment code_seg                ;Сегмент кода
start:                          ;Отсюда начинается выполнение программы
    mov ax,data_seg             ;\
    mov ds,ax                   ;/ Инициализация регистра DS

    mov ah,09h                  ;\
    mov dx,hello                ; > Вывод строки
    int 21h                     ;/

    mov ax,eseg                 ;\
    mov es,ax                   ;/ Инициализация регистра ES
    mov al,[cs:start]           ;Чтение байта, с которого начинается код
    cmp al,0xB8
    jnz exit
    mov word[es:0000h],1234h    ;Запись значения в сегмент eseg

exit:
    mov ax,4C00h                ;\
    int 21h                     ;/ Завершение программы

;---------------------------------------------------------------------
segment eseg
    rw 1                        ;Зарезервировать 1 слово
```

Дальние переходы, вызовы процедур и возвраты

Дальними (*far*) называются переходы в другой сегмент кода. При их выполнении меняется содержимое регистра cs. Они могут только безусловными. Ближние (*near*) переходы осуществляются в пределах одного сегмента. Аналогично есть дальние и ближние вызовы процедур, а также дальние и ближние возвраты.

Команда дальнего вызова процедуры сохраняет в стек не только *ip*, но и *cs*, чтобы можно было вернуться в текущий сегмент кода. Команда RET является синонимом ближнего возврата RETN. Дальний возврат осуществляется командой RETF. Она восстанавливает из стека регистры *ip* и *cs*.

Для наглядности пример:

```
format MZ                       ;Исполняемый файл DOS EXE (MZ EXE)
entry seg1:start                ;Точка входа

;---------------------------------------------------------------------
segment seg1                    ;Сегмент первый
hello db 'Hello, asmworld!$'    ;Строка
```

```
 6 hello db 'Hello, asmworld!$'      ;Строка
 7
 8 start:                            ;Отсюда начинается выполнение программы
 9     push cs                       ;\
10     pop ds                        ;/ Инициализация регистра DS
11     jmp seg2:do_it
12
13 exit:
14     mov ah,08h
15     int 21h
16     mov ax,4C00h                  ;\
17     int 21h                       ;/ Завершение программы
18
19 ;---------------------------------------------------------------------------
20 segment seg2                      ;Сегмент второй
21 do_it:
22     mov dx,hello                  ;DX = СМЕЩЕНИЕ строки в seg1
23     call seg3:print_str           ;Дальний вызов (cs,ip в стек)
24     jmp seg1:exit                 ;Дальний переход
25
26 ;---------------------------------------------------------------------------
27 segment seg3
28 ; Дальняя процедура для вывода строки (ds:dx = адрес строки)
29 print_str:
30     mov ah,09h
31     int 21h
32     retf                          ;Дальний возврат (восстанавливает ip,cs)
```

Программа состоит из трёх сегментов. Сначала выполняется переход во второй, затем вызов процедуры в третьем сегменте. Кстати, сегмент может содержать код и данные вместе — я поместил строку в начало первого сегмента.

Упражнение

Напишите программу, которая сравнивает две переменные и выполняет переход в другой сегмент в зависимости от результата сравнения. Если меньше, переход в сегмент 1. Если больше — в сегмент 2. Иначе в сегмент 3.

Комментарии:

plan4ik
14-04-2011 14:12

(JXX — conditions) не хотят прыгать в соседние сегменты … почему ?
Нужно бы написать статью по условным дерективам …. 😉
или я просто неправильно написал или недописал

```
format MZ
entry code_seg:Start

macro dosfn_ah [n_func]
{
mov ah, n_func
int 21h
}

segment data_seg
var1 db '7'
```

```
less_str db 0ah, 0dh, ‘less’, 0dh, 0ah, ‘$’
great_str db 0ah, 0dh, ‘great’, 0dh, 0ah, ‘$’
segment code_seg
Start:

push data_seg
pop ds

dosfn_ah 0x1 ; input char
cmp al, [var1]
jz .equ ; ‘jz’ neho4et prigat za predeli segmenta ((
js .less ; NUJNI uslovnii derectivi
jmp isgreat_seg:0
.equ:
jmp equ_seg:eStart
.less:
jmp less_seg:0

.Exit:
dosfn_ah 0x8, 0x4c

;_____
segment isgreat_seg
mov dx, great_str
dosfn_ah 0x9
jmp code_seg:Start.Exit

;_____
segment less_seg
mov dx, less_str
dosfn_ah 0x9
jmp code_seg:Start.Exit

;_____
segment equ_seg
text db 0dh, 0ah, ‘equal!’, 0dh, 0ah, ‘$’
eStart:
push ds
push equ_seg
pop ds

mov dx, text
dosfn_ah 0x9

pop ds
jmp code_seg:Start.Exit
```

спасибос за очередной урок!

[Ответить]

xrnd
15-04-2011 00:08

(JXX – conditions) не хотят прыгать в соседние сегменты … почему ?

Нет такой машинной команды. Для JMP есть ближний и дальний вариант команды, а для JXX — только ближний. В этом и прикол упражнения.

Условные директивы — это как аналог условных операторов if /else в других языках?
Лично я их не очень люблю — не всегда эффективный код генерируют.
Но написать можно. Если не ошибаюсь, в FASMe они реализованы в виде макросов.

Теперь о твоей программе. Всё правильно, суть ты понял.
Макрос dosfn_ah довольно удачный и используется уместно.
Вот только непонятно, с каким кодом завершится процесс.
При вызове функции DOS 4Ch в AL должен быть код 0, если это нормальное завершение процесса.

[Ответить]

plan4ik
15-04-2011 09:06

надо было обнулить al …

[Ответить]

annihilator
14-04-2011 20:50

пишу сюда, поскольку я только на 9 уроке, так больше вероятность получить ответ раньше.
у меня вопрос:
комманды ассемблера полностью соответствуют машинным командам или нет?
я в том смысле, существует ли избыточность кода программы в двоичном виде?
добавляет ли ассемблер лишние с точки зрения машинных кодов команды?

Вопрос поставлен с точки зрения простейших операторов а не программы вцелом. Понятно, что оптимизация программы вцелом зависит от программиста.

[Ответить]

plan4ik
14-04-2011 23:13

у каждых типов файлов есть свои форматы, например
*.com —http://ru.wikipedia.org/wiki/.COM
*.exe — http://ru.wikipedia.org/wiki/.EXE
выше описанные форматы генерирует компилятор вставляя в код информацию о виде (породы) файла
вот тебе инфа про PSP *.com файлов
http://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D1%84%D0%B8%D0%BA%D1%81_%D(

а команды это мнемоника опкодов и сточки зрения асма комманды и коды это одно и тоже — http://asmworld.ru/uchebnyj-kurs/sozdanie-listinga-v-fasm/

а все остальные байты твоей программы (несчитая генерируемый код формата) полностью совпадают твоим коммандам \ данным …

[Ответить]

xrnd
15-04-2011 00:13

Команды в программе точно соответствуют машинным командам, которые будут в исполняемом файле! В этом легко убедиться, если открыть программу в отладчике, дизассемблере или посмотреть листинг.

А вот директивы и макросы могут генерировать несколько команд или не генерировать ни одной.

[Ответить]

diger
18-04-2011 15:51

А как можно плучить доступ к памяти > 1 мб в реальном (ну или защищённом) режиме ?

[Ответить]

plan4ik
18-04-2011 19:26

v realnom nikak)))
http://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D

a v zashishonom sam poka neznau))) toje volnuet etot vopros takshto milosti proshu v topik
http://forum.asmworld.ru/viewtopic.php?f=3&t=128

[Ответить]

xrnd
19-04-2011 14:22

В DOS для этого используются специальные расширители (например http://ru.wikipedia.org/wiki/DOS/4GW), к которым можно обращаться через DPMI: http://ru.wikipedia.org/wiki/DPMI

[Ответить]

diger
18-04-2011 16:10

и кстати как работает директива stack ?

[Ответить]

plan4ik
18-04-2011 19:19

stack … eto obi4niy segment programno organizovaniy, u kajdogo segmenta est deskriptor a u descriptor'a est pole dostupa opisivaushiy tip dostupa (zapisi\4tenia, napravlenia rosta i t.d.) segmenta naprimer CS, DS, SS … tak esli etot tip imeet zna4enie rosta (D == 1) i zapisi (W == 1) to evo rost rastet v storonu mladshih adresov, s vozmojnostu zapisi, ostalnoe doljno bit poniatney …
Ja malo 4to ob etom znau … takshto sdes mojet bit sploshnie neto4nosti !

P.S: Popravte pojalusta esli eto rabotaet po drugomu 😉

[Ответить]

xrnd
19-04-2011 14:26

Директива изменяет значения в заголовке файла MZ EXE.
В зависимости от этих значений выделяется память для стека при загрузке файла в память.

[Ответить]

Гость
27-04-2011 10:08

Есть нормальная IDE для FASM-а?

[Ответить]

xrnd
06-05-2011 00:12

Вроде RADAsm — хорошая IDE, я раньше пользовался.

[Ответить]

Гость
06-05-2011 16:47

Смотрел мне не понравилось, давайте сами сделаем. Я уже начал писать не Delphi. Осталось доделать выпадающий список. Кто хочет помочь, пишите на Researcher86@Mail.ru.

[Ответить]

xrnd
12-05-2011 18:35

Могу пропиарить на сайте, когда будет готова.

[Ответить]

Гость
21-06-2011 08:42

Здравствуйте! Зацените самомодифицирующийся код.

```
format PE Console 4.0

entry Start

include '\Fasm\Include\win32a.inc'

section '.text' code readable writeable executable ; writeable

Start:

invoke SetConsoleTitleA, conTitle
e:
test eax, eax ; 85C0 test eax, eax
```

```
jz Exit
invoke GetStdHandle, [STD_OUTP_HNDL]
mov [hStdOut], eax

invoke GetStdHandle, [STD_INP_HNDL]
mov [hStdIn], eax

invoke WriteConsoleA, [hStdOut], mes, mesLen, chrsWritten, 0

call run
mov [run + 8], byte 0E8h ; 0C0h add -> 0E8h sub
call run

mov [e], byte 33h ; 33C0 xor eax, eax
jmp e

invoke ReadConsoleA, [hStdIn], readBuf, 1, chrsRead, 0
Exit:
invoke ExitProcess, 0

section '.data' data readable writeable
conTitle db 'Console', 0
mes db 'Hello, world!', 0dh, 0ah, 0
mesLen = $-mes
hStdIn dd 0
hStdOut dd 0
chrsRead dd 0
chrsWritten dd 0
STD_INP_HNDL dd -10
STD_OUTP_HNDL dd -11

run db 33h, 0C0h ;XOR EAX, EAX
db 0B8h, 15h, 00h, 00h, 00h ;MOV EAX, 15h
db 83h, 0C0h, 10h ;ADD EAX, 10h
db 0C3h ;RETN

section '.bss' readable writeable
readBuf db ?
section '.idata' import data readable

library kernel,'KERNEL32.DLL'

import kernel,\
SetConsoleTitleA,'SetConsoleTitleA',\
GetStdHandle,'GetStdHandle',\
WriteConsoleA,'WriteConsoleA',\
ReadConsoleA,'ReadConsoleA',\
ExitProcess,'ExitProcess'
```

У меня два вопроса:
1) Есть ли нормальный отладчик для программ Win64?
2) IDE for FASM готова для тестирования. Как положить её на ваш сайт?

[Ответить]

xrnd
23-06-2011 19:27

Хороший пример. Но ввод с консоли тут не работает.

1) Гугл сказал WinDbg x64. Лично я им не пользовался.
2) Круто 🙂 Отправь на мой почтовый ящик, я выложу на сайт.

[Ответить]

Гость
24-06-2011 13:13

:)))))

[Ответить]

T86
28-06-2011 17:08

Закинул.

[Ответить]

Cross
08-07-2011 18:32

а больше статей не будет?

[Ответить]

fufel
05-09-2011 21:48

Здравствуйте!

```
format MZ
entry seg_code:start
;_________________
segment seg_dat
a db 5
b db 4
;_________________
segment seg_code
start:
mov ax,seg_dat
mov ds,ax
mov al,[a]
mov ah,[b]
cmp al,ah
jl segm1
jg segm2
jmp seg3: S_1
segm1:
jmp seg1:S_2
segm2:
```

```
jmp seg2:S_3
;————————————-
segment seg1
S_1:
mov ah,08h
int 21h
mov ax,4c00h
int 21h
;————————————-
segment seg2
S_2:
mov ah,01h
int 21h
cmp al,0dh
je exit
jmp S_2
exit:
mov ax,4c00h
int 21h
;————————————
segment seg3
S_3:
mov ax,4c00h
int 21h
```

[Ответить]

xrnd
17-09-2011 21:53

Да, код правильный. Много сегментов получилось 🙂

[Ответить]

murashcour
27-09-2011 00:21

Буквально пролистал все вши уроки яинтересуюсь асамблером давно прочитал несколько похожих трудов в том числе и Калашникова сложно все воспринимается к сожелению не могу ни как разобрался с реализацией тригонометрических и логарифмических функций на асамблере буду очень блогадарен если вы поднимете эту тему желательно в консоле попозже если будет время попробую написать вам и порешать ваши задания спасибо вам и сайту CRACKL@B что попал на вашу страничку многое стало более понятно и систематизировалось в голове

[Ответить]

xrnd
30-09-2011 01:12

Тригонометрические и логарифмические функции можно вычислить с использованием сопроцессора.
Или интересует эмуляция сопроцессора обычными командами?

[Ответить]

andrew.NET
10-06-2012 16:51

А когда будет продолжение?

[Ответить]

Дима
21-07-2013 20:08

xrnd, спасибо тебе большое за этот хоть и не полный, но учебник по ассемблеру. Я уже полгода наталкиваюсь на то, что не могу нормально выучить хотя бы основы программирования на FASM, а тут нашёл твой учебник и уже можно сказать готов к бою! Но мне нужно попросить у тебя помощи: как я прочитал на главной страничке сайта, ты очень хорошо разбираешься в программировании под ‘голое железо’. Мне нужна твоя помощь, так как я в этом деле новичок (программировал только на pascal, C++ и теперь FASM). Мне нужно написать загрузочную программу операционной системы. Но ни то, как её записать на бут-сектор диска, ни то, как она собственно будет переносить данные с диска в оперативную память, я не знаю.
Если ты поможешь мне хоть чуть — чуть продвинуться, я буду тебе очень благодарен.
Заранее спасибо!

[Ответить]

anonym
19-02-2018 18:21

MBR (загрузочный раздел) — первые 512 байт диска.

[Ответить]

Владимир
19-05-2014 18:12

Огромное спасибо автору за труд!
Для новичка в изучении ассемблера этот сайт просто находка.
И продолжать печатать новые учебные материалы здесь просто необходимо.
xrnd ???

[Ответить]

lepota
05-11-2015 12:54

Где ссылка на 32 часть?

[Ответить]

Евгений
15-09-2016 09:04

Продолжение, нужно продолжить цикл !!!!

[Ответить]

Евгений
15-09-2016 09:05

Очень интересует работа с файлами.

[Ответить]

Aleksey
28-01-2017 23:11

Здравствуйте.
Не планируете ли рассказать про protected mode и основы OSестроения ?

[Ответить]

SOLOMON243
13-03-2018 23:31

Отличная работа!

[Ответить]

Ваш комментарий

Имя *

Почта (скрыта) *

Сайт

Добавить

Уведомить меня о новых комментариях по email.

Уведомлять меня о новых записях почтой.