

Глава 1

Организация ЭВМ на уровне ассемблера

1.1 Лекция 1

Замечание. *Максим Викторович Баклановский.*

Контакты: **baklanovsky**: skype, @mail.ru, @gmail.com

Варианты организации работы:

1. Загрузочная флешка.
2. Эмулятор(w32).
3. VM (Virtual Box).

Замечание. *Для эмулятора нужно попросить USBDOS.zip (по skype).*

Оффтоп. *В будущем нужно будет сделать доклад по настоящим спецификациям. Intel. Гук - 9 ссылок. По легальным ресурсам. Длинная арифметика. Ларри Уолл - создатель Perl.*

1.1.1 Работа с эмулятором

Разархивируем в C. NC далее. NC.bat его на рабочий стол(там пути). MAX каталог. Источники информации Shift + F1 = Norton Guide(Help по asm). Options DB = Подгружает базы. Справочник программиста. (Лукач, Гук).

Ralf Brown - сбор информации, перерабатывает. Файл с ошибками процессоров. Каталог Ints с прерываниями (F4) - просмотреть. Cntl + B = функциональные кнопки. MultiEdit - редактор. (Alt + F4 = альтернативный редактор, F2 тут вызывате меню, Alt + F1 = список окон). F2 - свою меню в Norton Commander.

LU => IO.DOC читать тут.

Util - утилиты. peroges.com резидент - программа на постоянное присутствие. "Взять вектора"и получают периодически .

TSR - terminate and state resident. Можно посмотреть, где сидит Norton. Когда что-то из Нортонa вызвали, то вся память у него, то он себя отрезает, ограничивая свой блок. Сюда загружается именно COMMAND.COM. Далее - свободная память, куда будут помещены наши программы.

NG => Лукач => undocumented.

Основные задачи - обработать прерывание с клавиатуры.

"perores"показал нам память(dump). Можно узнать, где лежит наш резидент. Нужно найти в память своей программы.

Turbo ASM, macroASM. MASM. Ассемблеры, где TASM от Borland, MASM от Microsoft.

DOS => Work => Shift + F4 = новый файл. Пробуем написать код.

model tiny ; чтоб не было сегментов.

.code

ДЗ. *org 100h ; Смещение в 256 байт, где хранятся параметры, а адресация сегмента кода, начинается с ; 100h*

here: jmp start

m1 db 'Hello, ASM!', 13, 10, 'доллар'

start:

mov dx, offset m1

mov ah, 9; когда вызываем прерывание, то передается управление библиотечной функции по адресу 21h

int 21h ; способ оформления библиотечной функции. MS-DOS call, который смотрит параметр из ah

ret ; - почему процессорная(ассемблерная) команда ret завершает исполнение программы. ; по сути ret возвращает управление, т.е. восстанавливает значение регистра команд их стека.

end here

далее Shift + F1, print string - ему соответствуте int 21h прерывание + 9 в ah, а то, адрес того, что мы выводим должен помещаться в dx.

Вспомнить Рюриковича - сегментная адресация. F10 - выход их NG.

Сохранили и создаем объектный файл с именем "1"

tams /m 1

Замечание. */? - узнать ключи. t - кол-во проходов. Написать программу, на которой ASM сделает столько проходов, сколько мы хотим.*

Проходим линковщиком

tlink /x/t 1

.COM файл, который в отличие от .EXE файла имеет ограничения на размер(64 КБ). Как-то связано с 100h.

Сделаем резидент программу 2.com - на постоянное присутствие в памяти. Примером резидент-программы является perores. menu => mem (тут можно его увидеть).

model tiny

.code

org 100h

m1: jmp start

m2: db 'Hello, I am TSR', 13, 10, 'доллар'; shift F1 ASCII char

start:

mov dx, offset m2; смещение считаем с нуля.

mov ah, 9 ;

int 21h ; передает управление на библиотеку

mov dx, 0 ; пишем и выходим. потом 12h

mov ax, 3100h ;

int 21h ;

end m1

ДЗ. Найти перерывы это место. Можно искать по тем символам-словам, которые есть в программе.

1.2 Лекция 2

1.2.1 Прерывание. Таймер

Все началось с того, что определить версию процессора Intel не такая уж и простая задача. Но с появлением команды **cuid** проблема решилась. Но в процессорах до Pentium этой функции не было, поэтому приходилось что-то придумывать.

Поступил вопрос: "Как посмотреть, где находится программа во время выполнения?". Как посмотреть это место, не делая её программой резидентом? Из всех предложенных вариантов больше всего подошел вариант с breakpoint'ами. Но посчитали, что вставив в код прерывание сделаем лучше.

```
xor ax, ax;  
int 16h ; прерывание bios по обслуживанию клавиатуры
```

Вообще прерывания бывают (это не полный список, как я понял):

1. Программные
2. Внутренние
3. Аппаратные

Заметим, что аппаратные прерывания зависят от таймера. Таймер зависит от системного времени. Тест был в том, что мы написали hlt (тем самым остановив процессор), но так как аппаратный таймер производит отсчет 18.2 раза в секунду, то он практически сразу же "завел" процессор.

Вообще есть Real Time Clock который обеспечивает переключение процессов в многозадачных системах, причем там таймер делает отсчет 1024 раза в секунду. Поэтому квант процессорного времени приблизительно равен 1 миллисекунде.

1.2.2 Управление. Отладчик

Передать управление, есть не что иное, как перейти к исполнению другой команды. Помним, что за это отвечает регистр IP = Instruction Pointer. В чем идея? Когда одна подпрограмма вызывает другую, то адрес возврата помещается в stack, причем когда вызываемая программа завершает работу, то она ret'ом загружает из stack'a в IP адрес вызывающей подпрограммы.

Отметим, что int param; - где param по сути означает смещение, которое мы отсупаем, чтобы попасть на нужную библиотечную функцию. По этому адресу как раз и передается управление.

Заметим, что когда вызываем tsr, то там вложено все вызывается. pc => command.com => tsr. Но tsr знает, что после того, как он отработает, он вернет управление и закончит работу, поэтому в памяти он себя не прописывает. Command.com на самом деле тоже возвращает управление (т.е. меняет регистр IP), но он все ещё отображается, поэтому это искажает наше представление.

Замечание. Передать управление - делать переход на область памяти

ДЗ. $int\ x \Rightarrow CD / X$ (такое представление). Намек на то, что алфавит *OPCODE*'ов переполнен. Почему для *breakpoint*'а *int* 3 выделили отдельный байтовый опкод *CC*?

Замечание. Если мы видим, что первый байт, в 2-х байтной связке для *OPCODE* равен *OF*, то это нам говорит о том, что мы должны смотреть в другой таблице команд.

ОС всегда завершает программы, поэтому мы можем увидеть в начале нашей программы символы **CD 20**, что говорить нам о том, что вызывалось прерывание *int* 20h (прервать программу = terminate programme).

Ещё раз вспоминаем, что *ret* восстанавливает в *IP* адрес из *stack*.

Посмотрели в памяти, что отступ, который мы прописывал *orh 100h*, использовался для размещения параметров. Заметим, что это смещение учитывалось, когда мы писали *offset m1*.

Интересная вещь, то то, что мы объявляли строку, как набор 1-байтных элементов, поэтому и писали "db"(data byte). Чтобы взять не 1 символ, а 2, например, нужно будет указать "word ptr m1 где word подразумевает 2 байта.

1.3 Лекция 3

Организационные вопросы: дополнительная пара для проверки домашнего задания.

ДЗ. Задача: написать свой примитивный отладчик. 2 момента: реализовать трассировку и *breakpoint*'ы. Загрузить файл - будет проблемно. Выполняем функции операционной системы.

Замечание. *MS-DOS*: Тим Патерсон. *DR-DOS*: Гарри Килдалл (король 8-битных операционнок). Ему *IBM* предлагали контракт, но он отказался, не согласившись подписать *NDA*.

Как открыть файл? *MS-DOS* переняла наследие от других ОС. Фредерик Брукс, руководивший разработкой первой ОС в современном смысле этого слова - *IBM OS/360*, получил Тьюринговскую премию. *Shift + F1* - вызов справочной системы, там же искать функции для работы с файлами.

OF(15) и рядом - устаревшие функции. *File* - ящик в картотеке, отсюда и *handle* - ручка этого ящика. Современные функции используют *handle* в качестве однозначного идентификатора открытого файла. Старый набор функций вынуждал хранить о файле значительно больше информации, чтобы открыть файл нужно было "описать ящик". Используем функции с *3Dh* (60-го) номера. Есть основные *handle*: *stdin*, *stdout*, *stderr* (*handle* - это просто целое число). Обращаем внимание на зависимость от *CF* (для возврата информации об ошибке). *JNC* (*jump no carry*) удобная команда. Перекидываем *handle* в *BX* (посмотрели в *close*). Нужен будет *READ*, чтобы считать *.com-файл в память. Поиск длины файла: *lseek* (66-я функция) с конца и с нулевым смещением. Одно из возвращаемых этой функцией значение - текущая позиция в файле. Её и следует трактовать как размер файла в байтах. Эрик Рэймонд. После открытия нужно делать *seek* 3-го типа. Так узнаем длину файла. Можно длину указать (*FF*). Из диска - в память загрузились и больше не читаем. Нужно обработать все завершения работы. Ограничения: отлаживаемая программа .COM и завершается *get'm*. Сам отладчик .COM.

Понадобится стековый фокус. Прерывания - 2е дело (внутреннее дело).

//

Работаем в debugger'е. Прыжок на 107, но 107 нет. в 108 90 - универсальная затиралка. Когда мы не ставим ключ /m => получили "пор" (Нет операции - такую инструкцию процессор успешно выбирает из памяти, декодирует, но ничего не делает. Только инкрементирует ip и переходит к следующей инструкции). Причина в том, что по умолчанию ассемблер совершает только один проход и за один проход ему не удаётся сгенерировать файл минимально возможного размера - из нескольких вариантов инструкций ему иногда приходится консервативно выбирать самый длинный. В некоторых случаях это приводит к тому, что ассемблер заменяет команду на более короткую, а свободные байты "забивает" пор-ами. Например, в 16-битном режиме существует как минимум два варианта безусловного перехода jmp: E9 + 2 байта на целое со знаком значение смещения, EB + 1 байт (для тех же целей). Если ассемблеру разрешён только один проход, он не может предполагать, что 1-байтового смещения окажется достаточно и резервирует под инструкцию jmp 3 байта. Если впоследствии окажется, что достаточно короткой инструкции, именно она и будет в итоге помещена в исполняемый файл. На место "лишнего" байта будет записан пор. Просто удалить лишний байт нельзя, так как изменятся смещения всех адресов и меток, расположенных ниже по тексту ассемблерного файла. Поправить их без лишнего прохода не получится.

С ключом /m ассемблер будет выполнять дополнительные проходы до тех пор, пока удаётся сокращать размер генерируемого исполняемого файла.

ДЗ. Задача - сделать столько проходов, сколько мы задаем.

Дизассемблирует все подряд. Процессор не понимает разницу между данными и инструкциями. Идет по логике линейного ассемблера. Для того, чтобы спозиционировать окно TurboDebugger (td) с дизассемблированным кодом в любую удобную позицию, можно воспользоваться хоткеем Ctrl+G и ввести нужный адрес.

Замечание. Говард Эйкен. Алан Тьюринг. 2 разных типа архитектуры: Гарвардская (адресные пространства кода и данных разделены), Принстонская (код и данные находятся в одном и том же адресном пространстве и неотличимы друг от друга).

0 в стеке положил DOS, чтобы ret'm перейти в начало сегмента и PSP, где записаны байты CD 20 (int 20h - завершение программы). Что лежит в стэке, чтобы мы вернулись. call делает call и кладет следующий за собой адрес.

В начале сегмента лежит наша программа.

ДЗ. Нужно стэком убить свой код. Чтобы стэк дошел до наших инструкций и затер код.

Golf - решить задачу, уменьшив размер программы.

Комментарий к функции h4 печати 16-ричного значения из файла 5.asm: ret - один, но возвращать он будет нас в разные места.

Как смотреть результат работы ассемблера без дебагера? Получаем листинг программы (tasm /m/1 5.asm).shl ax, 4 => разваливалась в 4 команды shr ax,1. Это некруто, так как ассемблер сделал что-то по своему усмотрению, не уведомив программиста. Написали .486 - разрешили использовать команды 486 процессора. Теперь, посмотрев листинг, видим, что у нас все заменилось на одну 3-х байтовую команду.

1.3.1 Пишем прерывания

Использование программных прерываний. Пишем свой код. Пишем свою функцию для вывода строки.

Всего есть 256 векторов, это просто адреса обработчиков соответствующих прерываний, организованные в таблицу, расположенную по самым младшим адресам в памяти. Номер любого прерывания, таким образом - просто индекс в этой таблице. Младшие 8 - (0-7) процессорные исключения. Пример: 0-е исключение - это деление на 0. Процессор не может выполнить команду деления на ноль и с помощью механизма исключений (исключение - особый вид прерывания, значит, у него может быть обработчик, как у любого другого прерывания) делегирует решение этой проблемы программисту. Следующие 8 штук (8-F) аппаратные int'ы первого контроллера, 8 - таймер, 9 - клавиатура. (10h - 1Fh) BIOS'а прерывания, 13 - работа с диском, 16 - клавиатура. (20h - 2Fh) забрал себе DOS (21 - функции дос, 22, 23, 24 - некоторые хоткеи (вроде принудительного завершения текущей программы), 28 - недокументированное прерывание, содержащее функции, полезные для создания резидентных программ). Итого: 48 номеров. Посмотреть в Brown'e что осталось для пользователей. Это от (F0-FF) зарезервировано изначально.

iret выталкивает из стека 3 слова, а не одно, как обычный ret. iret - это способ выйти из обработчика прерывания, нужные значения в стек помещает инструкция int: IP, CS, FLAGS. 25, 35 поставить вектор, прочитав текущий вектор. ds в .com-программах всегда равен cs и ss. Значение es не фиксируется, следует это помнить.

Делаем диспетчеры и вызываем разные функции. Shift + F3 = убрать выделение в редакторе ps. Написали разные функции, вызываемые в зависимости от нашего условия.

Гарри Киллдал. Создавал массив точек входа. Можно все тесты убрать. Диспетчеризация в начале. В самих функциях ничего не пишем.

mov dx, (offset start + 15)/16; расчёт размера программы в параграфах с верным округлением (необходимо для функции "завершить программу и оставить её резидентной в памяти).

Ошибка будет. Relative quantity illegal. Тут требуется какая-то типизация. Смещение не рассматривает как смещение. Нужно вычесть какое-нибудь другое смещение, например, offset _ = 100h, и добавить 100h.

Разделяли на клиентскую и резидентскую часть. Заполнилась колонка: взятые вектора (в отчёте утилиты TSR).

Посмотрели разработку сервисного разработчика прерываний.

ДЗ. Думать про задачу. Взять вектора: трассировка - первый int, breakpoint 3 - й int

Нужно грамотно проследить, куда возвращаться. Флаги для трассировки. Сначала перехватить int 1. Нужно отследить завершение программы.

1. Загрузить программу в память.
2. Сохранить регистры.
3. Установить флаги.
4. Печатать все IP.
5. Когда ret, должны выйти.
6. Breakpoint на IP.