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. pepores.com резидент - программа на постоянное присутствие. "Взять вектора"и получают переодически .

TSR - terminate and state resident. Можно посмотреть, где сидит Norton. Когда что-то из Нортона 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.

"pepores"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, offsset m1

mov ah, 9; когда вызываем прерывание, то передается управление бибилотчечной 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. m - 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 резидет-программы является pepores. 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 реpores't 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 появление команда **cpuid** 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 попсмотреть это место, не делая её программой резидентом? Из всех предложеложенных 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 прерывание 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 процессорновго времени приблизительно равени 1 милисекунде.

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 семещение, которое мы отсупаем, чтобы попасть на нужную библиотечную функцию. По этому адресу как раз и передается управление.

Заметим, что когда вызываем tsr, то там вложенно все вызывается. nc => 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 искажет наше предствление.

Замечание. *Передать управлениие - делать переход на область памяти*

ДЗ. *int x => CD | X (такое представление). Намек на то, что алфафит OPCODE'ов переполнен. Почему для breakpoint'a 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 операционок). 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 ошибки). 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 обработаться все завершения работы. Ограничения: отлаживаемая программа .COM и завершается ret'м. Сам отладчик .COM.

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

//

Работаем в debugger'e. Прыжок на 107, но 107 нет. в 108 90 - универсальная затиралка. Когда мы не ставим ключ /m => получили "nop"(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"nop-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 nop. 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 подрядр. 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 знаения 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/l 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'е 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, прочитать 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 nc. 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 разработичка 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.

1.4 Лекция 4

1.4.1 Выделение памяти

Загружает программу в себя, перехватывает прерывание, печатает трассу. Другой вариант - программа, в которой есть breakpoint.

Замечание. *Нужно ставить CC, а не int 3.*

Трассировка - пошаговое выполнение. Загружаем именно .com файл. Имя вызываемой программы - hardcode. Сначала написан сегмент дебагера, далее сегмент загруженной программы. Наблюдали, что сработало прерывание и напечатало нам строку. Трассировки подвергается каждый сегмент.

Выделение памяти? Alloc (или Malloc) выделение памяти в куче. Каждой запускаемой программе выделяется 4 Гб памяти. Т.е. мы можем обратиться к любому выделенному байту.

mov al, адрес

Говорим про виртуальной адресное пространство. Это поддерживается процессором. Процессор не проецирует адресное простарнство на физическую память. Только после запроса адреса процессор разбирается, в какой физической памяти лежит требуемая ячейка. При обращении к неспроецированной памяти - разница в выполнение - большая.

Обработчик 14-го прерывания. Появляется в режиме protected. Выделение памяти - проецирование запрашиваемой памяти на физичекскую. Но это не так. Нужно отдельное усилие, чтобы спроецировать память на физическую. Сброс происходит страницами по 4Кб. Квант сброса на диск.

Замечание. *Жестко привязать - (non-paging pool) некоторые связки виртуальной и физической памяти никогда не разрушаются. Запрос на пямять, который мы не можем удовлетворить. Должны заниматься сбросом*

Trash - для обслуживания обращения мы часто рвем одни и те же свзи, выделяя ресурсы. Алгоритмы не идеальны. Порой нет связи, которую можно порвать. Чем больше жесткой привязки, тем больше trash. Программисты решаю, когда использовать жесткую привязку памяти. С проблемой проецирования памяти столкнулся обработчик прерывания 14 (32 битный режим). Его нужно привязать жестко.

Есть таблицы для определения проекций, которыми пользуется процессор.

DDOS - Denial of Service. Всегда при установлении сокет в TCP/IP всегда немного выделяется в памяти non-paging pool. Поэтому возможна атака.

Оффтоп. *Защищаться от воков. Охотиться на волков.*

1.4.2 Выделение памяти в однозадачной системе

Как работатет DOS. Сидит внизу, выделяет память под PSP (Program Segment Prefix), загружает код программы. Потом выделяет всю память под нашу программу. Norton Guide сидел в памяти до того как мы загрузили нашу программу. NG активизировался, получив управление, но структура памяти не поменялась. Управление у Norton Guide, но у резидента только та память, которую он сначала себе отрезал.

Вся память у последнего исполняемого процесса. Он её обрезает, можно в получившейся свободной памяти запустить процесс. Теперь все оставшая память снова принадлежит последнему исполняемогу процессу. Уже смотрели то, как "отрезают"память.

Говорим про системные Alloc. Трехслойнай архитектура Alloc'ов. Процесс свободный, но должен ходить к ОС и отмечаться. "Я намерен обращаться к законно принадлежащему участку".

Триумвират - обсуждения alloc-malloc от Ромы, Артура и Баклановского.

Вызвыать функцию 4Bh - Load and Execute. Не в своем адресном пространстве.

Мы должны работать в своем адресном пространстве. Мы не режем новый кусок. Мы запускаем файл для трассировки в памяти, которою операционка считает истинное нашей. Пока мы не "резанулись"выделять нечего.

Пишем план:

1. Загрузить "файл 1"в память. Неприятности - выравнивание align. Начало PSP - program segment prefix. Нужно поставить за 100h. Взять нулевое смещение - это не получится. Сегмент может начаться, если у нас последний 0. Мы должны загружаться с выровненной точки

2. Передать управление "файлу 1".

3. Поймать возврат управления (позаботиться об это заранее).

4. Перехват / восстановление вектора V1.

5. Обработчик прерывания 1 - печать трассы.

6. Печаталка есть. Нужно аккуратно pop/push. Посмотреть, что происходит в современных компиляторах. Посмотреть пролог и эпилог. Есть процессорные инструкции. Организация работы со стеком.

7. Включить трассировку. Выключить трассировку.

8. Breakpoint - ставим, куда хотим, но дебагером. (CC - затирает, но нужно восстановить). Если мы хотим продолжить, то это нормально.

Главный пункт, взять и передать управление.

Поймать возврат - вещь из 2-х частей. По ret. Управление приходит в начало PSP, туда можно что-то поставить и в стек положить.

6-й включить/выключить trace flag. Одна инструкция.

Table 6-1. Исключения и прерывания.

Режимы работы процессора

1. Real Mode. 16-bit. Все стартует в Real Mode.

2. Protected Mode. 32-bit.

3. 64-bit Mode. Extended memory mode.

AMD перывае разработали 64-битную архитектуру. EM64T - Extended Memory 64 Technology - это расширение. Не новая архитектурная разработка. Появился этот 64-битный режим.

Замечание. *В Real Mode интересует первые 8 штук. Частый вопрос на экзамене.*

Инструкция UD - Undefined, придуманная для генерации exception'a.
Загрузили. Как передать управление.
в PSP -> куда файл залили.
CS => PSP IP => в 100h
retF CS:IP => пункт 3
jmp PSP : 0100h
call PSP : 100h
push cs
push ip
retf
Рома рассказывает про то, как поймать возврат.

1.5 Лекция 5

1.5.1 Совместное использование прерываний

Когда писали свое (0F1) прерывание, то мы не беспокоились о том, что его кто-то будет использовать, кроме нас. Прерывания int 1, int 3 - мы должны были восстановить, так как они используются всеми. Например, вспомнили, что в TD у нас не получалось пройтись и посмотреть, как ведет себя код, в котором мы перехватывали прерывания, вель, наверняка, TD тоже использовал в своей реализации int 1, с зажатым TF = 1(Traf Flag).

1.5.2 int 16

Смотрим, что 16-е прерывание у Norton Guide. Это можно посмотреть в menu(F2) => TSR, последняя колонка. Если будет 16 прерывание, то прерывание перехватит NG. NG передаст сервис DOS, DOS передаст BIOS, а потом обратно по цепочке нам должен будет поступать ответ.

ДЗ. *DOS/MU/kc 16. Нужно копировать воспроизвести ks в режимах 16 int и 9 int. Cntl + M - "нотка"(13-й) - внимательно обрабатывать. Связь кода и картинки*

Печатать на экран, что возвращает BIOS, (2 возвращенных кода и картинку).

Замечание. *Садимся на 16-й int. Мы не знаем, сколько обработчиков под нами.*

DOS перехватил 16-й int, направил на себя, потом DOS переправляет по старому адресу - на BIOS. Описываем распространение сигнала.

В начальный момент вверху сидит BIOS. То есть 16-й вектор, который находится в 16-й ячейке памяти, указывает на BIOS. После загрузки DOS менят 16-й int на себя. Теперь управление приходит в DOS, а DOS не имея нужной информации, передает управление в BIOS. "DOS сел сверху и передает управление по старому адресу". Потом "сверху сел"NG. NG "отдается"DOS. NG не знает ничего про BIOS. BIOS вернул DOS'у информацию, DOS протранслировал это уже NG'у. NG просыпается, но нам "наверх когда зовем int 16, то управление идет к нам, но NG уже подменил (Shift + F1 не печататется). Сняли NG - все печатается.

Схема совместного использования. Есть утилита "interrupt counter"считает кол-во вызовов 16 int. Его нужно отдать обязательно, т.к. это прерывание часто используется.

Work => symdeb -a (ассемблирование). ENTER выход их режима. -u режим дизасемблирования. Заметим, что вывод в littel endian. Байты в обратном порядке - соглашение о размещении многобайтных величин.

Посмотрим, что в ячейке, до того, как поменяем. В одном парагрфе 4 вектора, нужно пройти 5 полных параграфа и ещё 2 вектора. Посмотрели что в памяти, место где TSR. Конкретно смотрим pepores'ом. Забрали оттуда 25 AF 6A FA. К этим ячейкам памяти обращается сам процессор.

Запишем туда своё значение. Запустили код, 16-й int указывает на нас. Мы сели сверху. Смотрим ячейку. В TSR 3DA1 - это наш код. Смотрим 3DA1 - наш код, где в нашей программе мы ссылаемся на старые коды, которые мы сохранили, чтобы потом туда отдать управление.

iret возвращает старый флаги, который были у пользователя до вызова. Информация во флагах возвращается.

Замечание. *Когда на клавиатуре нажимается кнопка, то срабатывает аппартное прерывание, возможно, даже останавливается текущая программа, и срабатывает обработчик аппаратного прерывания.*

Процессор вызвал обработчик прерывания 9. Программа не заметила, что её прерывали и обслуживали клавиатуру. Для программы это невидимо, и она не знает, что что-то нажато. Программа узнает это через int 16.

400h - область данных BIOS. Буффер клавиатуры. Можем посмотреть, что там все меняется, если нажимать на разные кнопки.

Assembly => Low Memory Usage

9 int складывает все в буффер, а 16 int забирает данные из буффера. 16 программное прерывание, это по нашему требованию. Pepores просыпается по 9-му прерыванию. При alt alt alt ничего в буффер не кладем.

Оффтоп. *Можно приступать к реалзации int16.*

Способы организации

1. Синхронная организация (int 16)
2. Асинхронная организация (int 9)

Поллинг - прием постоянного опроса.

Как решается вопрос в **синхронной** схеме? Есть вычислительная задача. Но мы хотим работать с клавиатурой. Нужно модуль решающей проблемы с клавиатурой научить работать квантами. Далее необходимо организовать цикл в котором будем считать и опрашивать клавиатуру.

Как сделать это умно? Чтобы не делать лишних действий. Затраты вычислительного времени хотелось бы минимизровать.

Как это решается в **асинхронной** схеме? Вспоминаем многопочточное программирование. Пусть у нас есть модель T. Мы его считаем, но сначала написали перед ним KBD обработчик, и если получили int 9, анализирую вход можем приостановить вычислительный процесс.

В синхронном режиме все выполняется в ритме процессора, а в асинхронной схеме обработка постувших сигналов из "другого мира не зависим от ритма процессора. Другая временная логика.

В одном процессе формируются два thread'a: один под вычисления, один под опрос. Если поток во втором thread фиксирует нажачие, то идет "kill"на вычислительный модуль. Плюс в том, что не нужно менять код вычислительного модуля. Минус в том, что мы все равно делаем много лишних опросов. Традиционно такое решение относят к асинхронному режиму.

При обработке 9-го прерывания мы будем строить асинхронную схему, что при решении будущих задач вряд ли встретися в чистом виде.

Оффтоп. *USB - поллинговоре решение*

1.5.3 Int 9

Для части 2 необходимо 9-й вектор взять без DOS. Там есть проблема, которая покажет понимание идеи о том, что прерывание просходит в любой момент. Это значит, что нужно все сохранять и восстанавливать (включая флаги). DOS это int 21h, активен, только тогда, когда мы его зовет. Как его звать, ведь у нас однозадачная система? Как позвать второй раз DOS? В любой момент может быть прерывание, если в обработчике снова будет вызван DOS, то DOS рухнет. Когда стек переполнится, то он может случайно что-то затиреть. Побоялись переполнение, меняют нам стек.

Есть DOS, где-то в памяти - стек. DOS боится его переполнить. DOS берет ss, sp складывает их в ячейку памяти. SS устанавливают в свой системный стек (ss = x1, sp = y1) в ss, sp - новые значения. Когда DOS отработал, то из глобальной ячейки достаются оригиналы ss и sp, возвращая наш стек.

При втором вызове мы затираем глобальную ячейку. Второй вызов обслуживается хорошо, но когда возвращаемся на первый вызов - то нам уже не восстановить затертые ss и sp.

Ячеек глобальных несколько. Поэтому можем звать на разрешенных участках можем звать DOS второй раз. Генерируется Int 28 - когда позволено вызвать DOS второй раз, можем его перехватить и второй раз вызвать DOS. NG перехватывает 16 и 28. В результате, он по 16 int ставит флаг, что было бы хорошо проснуться. NG просыпается по 28 int.

int 28 очень много, поэтому для нас ожидани очередного int - незаметно.

1.5.4 Работы процессора

Сверху выборка, декодирование, снизу - выполение. Процессор получаетс байт EAh, понимает, что нужно выбрать ещё 4 байт, если CD то выбрать ещё 1 байт.

На этапе выполнения могут возникать exceptinon (деление на 0), на этапе декодирования могут возникать exception (undefined).

В области межну завершнием выполнения одной инструкции и началом декодирования другой может прооизойти внешние прерывание.

Выполнение одной инструкции

1. Out of order execution. Сюда включается компиляция в микрокод. Все, что послыем процессору компилируется в микрокод и выполняется в том порядке, который ОНИ определили.

2. Суперскалярность. Несколько конвейров для выполнения микроинструкций. Наприме, адресный конвейр. В процессоре есть специальный модуль сборки, который собирает результат. Уже на уровне процессора поддерживается виртуализация.

3. Спекулятивность. Выборка делается далеко заранее. Как вариант,из двух ветвлений выбирает одну ветку и по ней делает предвыборку. Или если будет статистика, то делаем так, как делали раньше. Все статистические данные хранятся в процессоре. Можно hint-ом подсказать, но не факт, что это даст прирост производительности.

Одна ножка, за которую процессор "дергают"внешние устройства. По дороге стоит контроллер(8 входом, 1 выход) - контроллер прерываний. IRQ - номера входов. Проекция IRQ на int: с 08h по Fh, 71h по 77h. На второй выход IRQ контроллера навесили каскадно ещё одни контроллер. У обоих контроллерах на 0 входах подключены часы.

Часы у первого контроллера = 18.2 Гц, системный таймер. Часы у второго контроллера = 1024 Гц, RTC (Real Time Clock). На RTC работает современная многозадачность.

Замечание. *Кен Олсон - создатель компании DEC. Предсказал много направление в области современных технологий. Дэйв Кутлер - создатель Windows NT. Билл Гейтс предсказывал эру мобильных вычислительных устройст, связанных в одну сеть.*

Контролееры назывались PIC - Programming Interrapt Controller. Сейчас APIC на 24 входа.

ДЗ. *Прочитать к следующему разу: LU/io(alt + F4) ; третий пункт, программируемый контроллер прерываний(21 страница); картики в таймере на int8*

В следующий раз обсуждаем, что происходит, когда сигнал от клавиатуры доходит до процессора. NMI - посмотреть.

1.6 Лекция 6

ДЗ. *Вопрос, почему после int 21h, не протрассировался ret. Такое проскакивание происходит и в начале.*

Причина, почему сделали однобайтную инструкцию CC, а не int 3 (CD 03) - мы, захватывая 2 байта, затираем другой байт. На этот байт мог быть jmp. А в случае однобайтной инструкции CC мы избегаем данной проблемы. Если в том байте - точка входа, то затирая что-то мы портим точку входа.

1.6.1 Обработка аппаратных прерываний

Восстанавливаем картинку с ножкой процессора, общающейся с программируемым контроллером прерываний(ПКП).

Как проходит сигнал. Произошло дейтсвие на клавиатуре. На клавиатуре есть чип, он обрабатывает то, какая клавиша была нажата. Кладет код в буффер чипа клавиатуры. Чип клавиатуры меняет напряжение на проводе, связывающем его и контроллер. Чип клавиатура дергает ПКП. Учитывая приоритет, выставленный на ПКП, дергаем процессор за ногу. Процессор доходит до момента, когда может происходить "внешнее прерывание". Процессор знает, что его дергают за ногу для внешних устройств. CLI(Interrupt Flag = 0) - не реагируем на прерывания, STI(Interrupt Flag = 1) - обрабатываем прерывание. Если IF = 1, то процессор не выполняет инструкцию по CS::IP, а начинает заниматься прерыванием. Во-первых, мы должны были сделать все невидимым. Должны сохранить регистры (кладем с stack): регистр флагов (который потом будет меняться, например, IF), CS, IP (чтобы потом вернуться). Это все складывается на stack сразу.

Далее процессору нужно узнать, какой брать вектор. Для этого он посылает запрос контроллеру, желая узнать номер контроллера.

Абстракция **in, out**. Это все, что связывет процессор с внешним миром. Все проходит через 2 эти инструкции. Адресация - пространство памяти(RAM в любой момент к любой ячейке памяти). У процессора 2 адресных пространства: mem, IO. IO - пространство входа-выхода, где 16 bit на адрес. Каждый байт в IO - это порт.

Замечание. *Посмотреть, что такое "грязная дюжина"IBM. Элитная команда. Разрабатывали ISA шину*

Процессор, обращаясь в IO broadcast'ом шлет адрес. Устройства знают, какие порты его. Устройство, приписанное к порту - понимает это. Процессор запрашивает с этого адреса in или дает на запись out.

Процессор узнает у контроллера IRQ. Информация для mapping'а отражения IRQ на int хардкодом прошита в процессор, выполняется процессором.

Процессор вычисляет адрес вектора, достает CS::IP и передает управление.

Замечание. *Как контроллер узнает, что процессор начал обрабатывать прерывание? Процессор сообщает ПКП, что начал им заниматься. ПКП теперь ждет и не "дергает"процессор.*

Мы и процессор: действий в секунду(10, 10^9), квант времени ($10^5 \approx$ сутки, 10−3).

Замечание. *Зачем при входе в обработчик сбрасывается IF, IF(0)*

Оффтоп. *Юрий Нестеренко. "Дай бог памяти". "Что было бы если программисты строили дома". "Проект генезис".*

Диаграмма(цикла обработки команды) - выборка из памяти и декодирование, выполнение. Промежуток, когда может произойти внешнее прерывание.

Порядок букв. Нужно учитывать это.

Вектор, это 4 байта. Значит можем смотреть в pepores. Не забыть перевернуть все.

ДЗ. *Вектор нужно перехватить вручную*

1.6.2 Практика

Как идет работа с клавиатурой. Есть обработчик прерывания 9. Есть обработчик int 16. Есть где-то в памяти KBD buff. Сигнал приходит int 9 вызывается, складывает данные в буффер. int 16 зовется из программы и забирает что-то из буффера. Мы будем ждать некую кнопку, как только обнаружим код нужной кнопки => тогда закроем программу.

В цикле будем проверять флаг. Если не нажата наша кнопка, то выполняем цикл. Занятое ожидание.

Адрес порта клавиатура 60h. На 60h отзывается чип в клавиатуре. Карта портов есть у Brown'a. in берет байт из буффера клавиатуры. Проектировщики клавиатуры решили, что мы сохраняем один байт по нажатию клавиатуры.

Код нажатия и код отпуская - 2 разных сигнала.

Может быть неизвестное значение регистра ds. Когда наш код проснется - предугадать невозможно. Сигнал, по сути, из другого времени. Асинхронный стиль. Поэтому можем писать cs::f9. ds по default поэтому мы его модифицировали. Можно всегда писать префиксы и быть уверенными во всем.

Когда мы обрабатывали только esc, то то, что сложилось в буффер и не считывалось 16 int потом считалось NC (который направил int 16).

Печатать DOS'м нельзя. 21 int в аппаратном обработчике - плохо, точно.

Если вызвать кого-нибудь, кто ещё работал с DOS, то смогли повесить DOS. Нужно провести эксперимент, чтобы повесить DOS.

int 10 - вывод на экран. Печатать можно BIOS'м.

ДЗ. *Нужно сделать нормальное решение - 10 int'ом печатать в колонку. Функция 6 или 7*

Оффтоп. *Как только появляется интерфейс, то программирование резко усложняется. DirectX - библиотека, её сделали по заказу Microsoft. Все геймерные производители перешли на DirectX.*

Нужно напечатать символ и сдвинуть курсор. Перед этим считать позицию курсора. Посмотреть NG. Что нужно будет. Нам нельзя будет отдавться старому обработчику.

Попробовать добиться того, чтобы коды приходилив обратном порядке. Обработчик должен быть прерван следующим сигналом. Должны сделать sti. По одной клавише 2 кода. Серые кнопки, начинающиеся с E0. sti нужно ставить сразу.

Распространение сигнала по железу, а потом распространение по обработчику.

Программируемый периферийный интерфейс - тут смотреть порт 60.

1.7 Лекция 7

1.7.1 Измерение времени

Как можем узнать время? 02Ch - функция DOS. Тикает с частотой 18.2. Спрашивае время у DOS'а. Мучительная процедура вычиитания т.к. различные основания для минут, секунд, часов. Можно прочтитать из памяти BIOS'а по адресу 046Ch. Можно воспользоваться int 8, который прочитает значение по адресу 046Ch. Это удобно, т.к. достаточно будет вычитать из одного значения другое, чтобы определить временную разницу.

Процессор меняет время в той ячейке, когда приходит 8 прерывание по IRQ 0. Если переполнение, то 0470h - выставляется Time Overflow Flag. После того, как появилось 8-е прерывание, то 8-е прерывание генерирует 01Ch. Ранее DOS рапространял информацию по через int 28. Зачем нужно дополнительное прерывание 01Ch. Когда закончили обрабатывать прерывание, должны сообщить контроллеру прерыванию об этом.

Если мы сначала перехватим int 8, в начале своего обработчика позовем старый, потом вернемся, то после этого момента все происходит 01Ch.

Перехватили 8-й

1. Подготовить стек
2. Вызвать старый обработчик
3. Далее 1Ch

Заметим, что 1Ch автоматом выполняет первые 2 действия. Мысль, что так будет удобнее.

Есть ещё Real Time Clock. Можно опираться на него, по int 70h спрашивать.

Можно спрашивать время у процессора, но это не всегда удобно.

Узнаем, что таймер 3-х канальный. 1 канал - важное. 0 канал - int 8. 2 канал - мы можем опрашивать.

Чем отличается статическая память от динамической. Оперативная память - динамическая. Статическая (кэш) имеет большую скорость. Статическая помнит. Динамическая забывает.

Для динамической памяти читают, перед тем, как все она забыла. Потом записывают снова. Рефреш оперативной памяти. Операцию производят для всей памяти.

Вот для регенерации динамической памяти необходим 0-й канал таймера. Это реализовано аппаратно. Это находится внутри микросхемы памяти. Таймер 8253 - здесь 3 таймера = 3 канала, на 0-м канале (int 8 = System time). 2 - канал не стартует сам по себе, он отдам нам.

1.7.2 Пример работы с документацией. Реализация временной задержки

Одно разовый проход по документации. Задачи - посторить временные задержки, меньшие, чем $\frac{1}{20}$ секунды.

Нужны ли задержки в программе? Да, например, чтобы правильно писать в устройство(работа с устройствами по спецификации). Для реализации работы с пользователем - писать интерфейсы. Также необходимо, когда пишем тесты.

В старых играх задержки зависили от частоты процессора (задержки были циклами). Поэтому на новых компьютерах все было очень быстро. Поэтому появились замедлители процессоров.

Сделаем нормальную задержку. Поиск на F7. Смотрим на картинки, ожидая момент, когда становится понятно. Нужно, чтобы счетчик делал dec(), пока не получим 0. 27 % документа io. В листинге по таймеру ошибка. Неправильное управляющее слово. Ошибка в последнем бите.

Составляем самостоятельно управляющее слово. Alt + F1 - переключение между окнами.

ДЗ. *Почему тикает int 8, когда не дали отбой int 9. Ответ искать в разделе ПКП (раздел 3). Искать приоритеты. Как запретить прохождение сигнала от конкретного устройства? На уровне контроллера. Можно заблокировать все сигналы.*

ДЗ. *Классификация прерываний от Intel. Нужно знать. INT 2 приходит снаружи на отдельную ногу. В эти exception'ы свалили сходии по природе вещи.*

NMI идет из за ошибки в памяти

ДЗ. *Найти как замаскировать NMI*

1.8 Лекция 8

1.8.1 Доклады

Нужно в мае подготовить доклад.

ДЗ. *Одна задача касается PCI. Нужно выдать коды PCI устройств. В следующий раз будем обсуждать PCI. Вторая задача - сделать свой загрузчик. Меняем boot сектор. Пишем свой и сами это складываем. И нужно с этого грузиться. Дополнительно к этой задаче нужно написать программу, которая будет показывать вектора и некоторую информацию.*

Обсуждаем темы. Все разделили на общеизвестные стандарты и процессорные команды. Темы разбирать будем в мае (или сейчас уже).

Memory <=> Процессор <=> I/O

Можно запрашивать у системы:

1. Порты (к I/O)
2. IRQ - дорога от устройства к осбобой ножке процессора
3. DMA - direct memory access. Устройства могут писать напрямую в память, не отвлекая процессор.
4. Память (memory)

Определение. *Шина - среда передачи сигнала.*

Системная шина или локальная - шина, которая реализует физическую связь процессора и памяти. IDE - процессорная плата на диске. ATA/ATAPI - (PIO для записи в процессор). Внешние устройства подключаются к процессору через PIO, передавая данные которые будут проходить и через системную шину.

Задача поддержки группы высокопроизводительных машин для мощных запусков. Постоянно были тесты железа. Появилась Win95. Жесткие диски стали работать в 2 раза быстрее. Удвоилась внешняя шина, обошли ограничения. Ultra DMA - могут появиться в лабораториях, но когда они добираются до пользователя - проходит время.

DMA предлагает прямой доступ к памяти. Процессор программирует DMA, говоря "что взять "куда положить". Когда DMA заканчивает, оно должно оповестить процессор через IRQ.

SCSI - делал сразу DMA. Все всегда упирается в цену вопроса. Кому критично, покупали SCSI. Ранее не могли принять стандарт по ATA. Для каких классов устройства важен DMA?

I/O - можно тоже классифицировать. Это можно сделать самостоятельно.

Как память на устройстве отображается на оперативную память.

С BIOS'ом общаемся через порты. Через порты мы можем делать update.

Проецирование на память. Запрос "хочу mov из памяти в регистр". Сначала идет запрос на адрес в память. Если запрос на BIOS память, то BIOS не откликается, а используется информация из памяти. Говорят, что "на память проецируются порты". Когда к BIOS'у обращаемся, как к памяти.

1.8.2 Boot

Холодный reset - полный набор действий. Первый раз за много время производим запуск. Кнопкой reset.

Горячий reset - на лету, быстрая перезагрузка.

Адрес:**FFFF:0;** g105 - холодный reset. Адрес по которому перезагружаемся. Все это можем писать в symdeb.

top500.org сайт о суперкомпьютерах Как стартует суперкомпьютер. Группа суперкомпьютеров подгатавливает к запуску этот суперкомпьютер. "Разогревает его".

Вообще изначально стартует одно ядро. Оно стартует в реальном моде. Горячий reset это int18.

Смотрим, как устроено все в памяти. После F000:0000 идет сам BIOS.

Итого получаем: вектора(1K), память BIOS(256b), DOS(до A000, т.е. 640K). Память на видеокарте проецируется на C000(32K). Можно посмотреть на неперехваченный int 10. Вообще это спроецировано, а значит - продублировано.

Вообще, смотрим кусок 0 - 1 Мб. Особенные программы идут с устройствами и проецируются на память в "середину".

Стандарта в области жестких дисков нет. Не могут прийти к соглашению уже лет 30. Это же происходит с видео и сетевыми картами.

Некоторая память может остаться свободной. 160K свободных. После половины C000 и до F000 располагается Upper Memory Block. Гарри Киллдел и его фирма Digital Research. DR-DOS Киллдела получил золотую медаль за лучшую ОС. Гейтс проиграл, но зимой он копирует DR-DOS в MS-DOS 5.1. Лицензионно чисто. Легче кодить, чем проектировать. Через 10 лет компания Киллдела перестала существовать.

Есть ещё HighMemory. Нехватает битовых комбинаций. Больше чем 2^20 адресовать нельзя. Когда 20-bit перестало быть физическим ограничениям, появились. 32-bit процессоры. Ошибка A20R.

Было 20 ножек для передачи адреса. Теперь 24 ножки или 32. Должны эмулировать 20 битный режим. Должны обнулить те, что не используем. Intel отрезали с 21, а не 20.

ДЗ. *Сколько памяти потеряли Intel(а нам добавило памяти)?*

Есть техника Родена. Есть неадрессуемые = теневые регистры. Когда из реального режимы в 32-bit'й, а потом обратно, то нам достаются под 16-битном режиме возможность использовать большую 32-bit'ю адресацию. Вопрос, как обратиться? Даст ответ на вопрос.

Штрихованный: "от B000 до BFFF"под видеопамять.

4000b на страницу. Чтобы быстро перемещаться между страницами - держим в памяти из 32K несколько страниц. Просмотр страниц.

Пишем на разные страницы. pn - резидент, показывающий страницы. dm - display memory

ДЗ. *Взять утилиты для работы с DOS*

ic - industrial computer

Как програмно определить, какой монитор? Ч.б или цветной? Артур говорил про 61h, 7 bit.

ДЗ. *Попробовать в symdeb - поменять форму курсора через порты.*

1.9 Лекция 9

Замечание. *Баклановский говорит, что на экзамене будет специально запутывать. Стоит смотреть, с чем соглашаться, а с чем нет.*

1.9.1 Видео

Смотрим разные решения для видео: mono - 2 цвета, CGA(Color)- 4 цвета, EGA(Extended), VGA(Video), SVGA(SuperVideo). GA - это Graphic Adapter(Graphical Array) - для нас в названиях нет особой разницы. VGA поддерживается современными устройствами. Ранее те, кто писал в порты адаптера, дублировали запись в память. Это необходимо, чтобы другие узнали информацию о видео-режиме, что мы сделали. Т.к. до VGA порты адаптера только на запись.

B8000 - адрес видео буфера. Дублирование принято из-за старых моделей. Достаточно в памяти поменять форму курсора. И курсор становится именно таким, каким нам нужно. С VGA пришло 256 цветов - это была революция. В EGA была подготовительная работа для VGA.

Пирамида абстракций. По X - возможности. По Y - абстракции. Идя вверх по абстракциям, мы лишаемся огромного количества возможностей.

Что ещё пишется в память - сами порты. 3D и 3B (посмотрели symdeb,-dw0:463 - первый байт 03D4).

Процессор как-то общается с устройством(in,out). В устройстве за этим реально стоит 2 ячейки, т.е. 2 порта. Один индексный, другой - порт данных. Внутри куча регистров и указатель. Вся куча регистров адресуется через указатель. Сначала говорим, какую ячейку хотим записать. Заполняем значением индекс. Потом передаем данные. По индексу записываем полученные данные. Шина - интерфейс, обертка (ISA, PCI).

Через порт 3D4 будем посылать адрес. Через 3D5 посылаем данные. Пишем в sumdeb:

```
mov dx, 3d4
mov al, a
out dx, al
inc dx
mov al, 1 ; вроде, так
out dx, al
dec dx
mov al, b
out dx, al
inc dx
mov al, .. ; что-то нужно положить
out dx, al ;
press enter
```

Сначала послали индекс, потом данные. Если кто-то влезет между нашими out, выполняющими данную задачу, то это может плохо закончиться. Нет атомарности, хотя действие должно проходить, как транзакция. Для этого, на данном этапе, можно исправить все, используя **cli, sti.**

Буквы адаптер рисовать не умеет. Весь экран разбит на ячейки 8x25. Ячейки состоят из пикселей. В прошлый раз мы заполняли коды. Каждому коду соответсовало заполнение. Таблица такая - знакогенератор. Появилась в EGA. В VGA можно и читать эту таблицу.

ДЗ. *Постотреть таблицу знакогенератора*

ASCII - соответствие кода и картинки. Знакогенератор - соответствие кода и заполнения ячейки пикселяеми. Проблема со шрифтами - они плохо масштабируются.

Курсор - это линии.

1.9.2 PCI

Little Endian: mov x, ax; в памяти получим x = al | ah. Именно так уложены. В al будет лежать адрес, в ah будут лежать данные. Одновременная передача индекса и данных. Избавились от возможной проблемой случайного прерывания.

```
mov dx, 3d4
mov ax, 10a
out dx, ax
```

mov ax, 100b
out dx, ax
g(последняя ячейка)

У PCI 32-х битные порты. Мы сразу принимает 4 байта. Так идут все операции.

Видео начало набирать мощь. Эффект белого листа. Экспериментально пытаются отличить лист бумаги от монитора. После 100 Гц. Люди перестают отличать. Получается ещё один класс устройств, которые захотел скоростной доступ к памяти. Видео уходит из I/O. Local Bus от фирмы Vesa. Intel предлодила PCI шину.

На шине между CPU и RAM ставили мезонин - bus. Вклиниваем шину. К этой шине стали присоединять монитор. Локальная шина - шина, имеющая непостредственный контакт с процессором и памятью. Рисунок с PCI шиной, которая соединяет много устройств(в центре PCI из неё идут пути к процессору, RAM, I/O). AGP шина идет в обход PCI, напрямую от видеокарты в RAM.

Картинка. MCH - Memory Control Hub(соединяет CPU, RAP, AGP4X, ICH2). ICH2 соединяет в сеть ATA, USB, PCI, LAN. Но для установления взаимодействия должен возникнуть канал. Канал должен администрировать процессор. Есть вариант с CPU, но это прошлый век. Должны быть отдельные процессоры, как мы смотрели процессор DMA для взаимодействия точка-точка память-диск. Сетевая технология - туннель поверх Hub.

Чипсет - реализация архитектуры.

Определение. *FCB - front size bus. Шина, напосредственно к процессору(от какого-нибуль switch)*

Intel Core уже имеент вместо 2-х хабов - отдин switch. Каналы стали динамическими.

На PCI не так много устройств можно посадить. Зато в PCI есть возможность делать еще шины PCI. Соединение связывающее PCI и PCI - называется PCI Bridge. Соединение 2-х разных сетей - bridge.

Адрессация географическая. Куда воткнули, отттуда и получился адрес. В устройстве не прописан адрес. Адрес в географии, там, куда подключаем. Структура адреса: Шина-Устройство-Функция-Регистры. Нумерации шин PCI. Устройство - номер. Внутри самого устройства поддерживаются нумерация функций. Все навороченные чипсеты поддерживают логическую адресацию PCI.

Пример функции: в самом чипсете несколько устройств определены в один девайс. Спецификация доступна по адресу PCISIG - тусич про PCI. Как RFC.

Ограничение на PCI 512 МБ/сек (недостяжимая) - максимальная частота. 256, 128. Зависит от разрядности шины.

Нужна будет фолксономия для решения усложненной задачи. Коды нужно будет превратить в строчку. Нужно текстовый файл превратить в удобный. Нужно проиндексировать. А потом искать.

47 страница документации. Configuration space.

Что было до PCI? На что претендует устройство? Устройство может претендовать на память, IRQ, порты, каналы DMA. Конфликт устройств может быть(могу претендовать на одни и те же наборы). Диспетчер должен выдвать, по идее. Устройство может иметь разные наборы претензий. Устройство никак не может узнать о том, занято ли, что нам нужно или нет. Поэтому нам предложиди jumper'ы. Plag _ Play - безумная надстройка. В PCI уже сразу был Plug_Play.

Холодный старт, потом POST, потом загрузка. На этапе POST происходят разводка портов. Формат адреса. Старший байт 80. На bus - 8 бит. device на одну bus - 5 бит. 8 функций на device.

ДЗ. *Комментарии по задаче. Нужно определить все PCI устройства и вывести их номера на экран. Для талантов - нужно составить соответствие номеров и имен и печать имена PCI устройств. Читаем по 4 байта из PCI. Multifunctional устройства нужно определить и отдельно поработать с ними. Перебираем функции устройства, только когда устройство multifunctional. Если VendorID = FFFF -> устройства нет. Если не FFFF, то печатать.*

1.10 Лекция 10

1.10.1 Архитектура

Процессор соединяется с "облачком с которым соединены другие устройства. Говорим, что есть 2 канала: канал адреса, канал данных. Пример аналогия из сетей: siplex, half-duplex, full duplex. Если все свести к одному канался - то это путь мультиплексирования. Нужно оценить все "за"и "против". Есть принципы фон Неймана: first draft. Тайминги инструкций современного Intel - таблица времен исполнения инструкций в тактах (Optimization manual, C). Некоторые инструкции исполняются за 0.3 или 0.5 тактов - внутренний параллелизм в процессоре. Никто не знает, за сколько выполнятся комады процессора. 4 млрд операций, 1 млрд тактов в секунду. Работа с памятью 10-16 Гбит/сек. Моучли и Эккарт - сконструировали "Эниак". Нейман опубликовал работу, подписанную только им, но описывающую работу целой группы. Нейман добавил от себя лично - от увлекался самомодифицирующими автоматами. Он продвинул идею того, что в памяти можно проводить модицикации памяти, чтобы получить самомодифицирующуюся программу.

Замечание. *Гарвардцкая архитектура - разделенная память данных, память для адресов. Принстонская архитектура - совместная память. Проблема buffer overflow.*

Способность модицикации программ - не увеличивает производительнотсть системы. Идея использования гарвардской архитектуры принесло с собой проблемы обращения с памятью и её модификация.

Принципы фон Неймана:

1. Принцип двоичного кодирования
2. Принцип адресуемость памяти. Адресуемость блоками. Линейный порядок на множестве адресов. Идея RAM.
3. Принцип программного управлвения. Инстркции в памяти. Инструкции выполняются друг за другом.
4. Принцип одродности памяти.

Смотрим шины: параллельные и последовательные. Internet - последовательные шины. Ethernet - последовательная шина. Кто быстрей? Необычный результат.

Раньше внутри PC были быстрей параллельные шины. Сейчас USB - последовательный. SATA последовательный.

ISA,

IBM PC/AT. Первая персоналка с HDD. Там появилось соединение ATA/IDE - идея обхода общей шины ISA, чтобы её не загружать. С видео - проблемы была решена проецированием в память. Работают все те же абстрацкии: через порты, через прерывания(ресурсы), через DMA, через память. Хитросплетение проводов. DOS не знает про USB, он посылает что-то в порты.

1.10.2 Программа PCI

Оффтоп. *S.Warrent "Hacker's delights советуют читать.*

Win-image. Создаем дискету(3-х дюймовую). PCI1.COM - перетаскиваем туда. IMA - образ. Выбираем в DOS дискету(образ IMA). Так, наверное, можно будет передать данные с именами.

Порты, IRQ, отображенна память, DMA. То, что ведет нас к устройствам, как программистов. Что мы сейчас делаем?

Записываем на HDD. Мы посылаем команды в контроллер диска. Это, все равно, использование по назначению.

Когда начинаем работать с конкретным PCI устройством, то снавала узнаем мета-данные. Мы выясняем, что это сетевая карта, product ID, vendor ID.

Облако можно уподобить Internet. Наша IP абстракция - это 4-ка.

PCI-specification. (p.215)Configuration space.

Проблема.

Представим, что на разных заводов сделали 2 разных завода. Но они не согласовали, какие использовать порты. Одновременно тогда с этими устройствами рабоать на сможем. Основная проблема - согласовать все с производителями аппаратур(OS 360). Vendor'ы не захотели понять требования для включения в ОС. Майкрософты сами переписывают драйвера. Зная о том, что vendor'ы не договорятся, каждый вендор учить устройство поддерживать несколько диапазонов портов. Включаем устройства. Конфликтуют, нет? Если да, то джамперами можно поправить конфликт. Это не поток не поставить.

Придумали plug и play, на шине нельзя перевести все устройства в состояние конфигурирования. Т.е. мы хотим конфигурировать устройство, а устройство думает, что мы работаем не с ним.

Как передать устройствам то, что мы в режиме конфигурации. У нас 2 разных дела: конфигурация иди типичная работа.

Замечание. *PMP - экзотика. Архитектурно не предусмотрен режим конфигурирования. Поэтому фигачили всевозможные костыли. Модуль ртр - не проблема. Нужно было предусмотерть семантику switch.*

В PCI режим конфигурирования полностью отделен от обычного режима. В нашей задаче мы получаем доступ к конфигурационным данным.

На всем этом фоне работают BIOS и устройства.

В следующий раз мы будем проходить от включения питания до запуска OS.

ДЗ. *Нужно готовить доклад, парень!*

A сегмент, B сегмент - графическая память (отображения с видео - занято). Смотрим C сегмент.

FFFF:0000 - главный вход в BIOS.

util -> mft (программа манифест)

В сегменте F000:0000 сидит главный BIOS. После того, как он свои 64к обработал, то смотрим, какие ещё есть устройства(дополнительные).

55 AA - нужно искать пометку, что это BIOS устройства.(55 AA) просто нравилось. Начиная с C000:0000 начинаем смотреть до F000:0000 по 4к.

В папке LU => bios. Роспись AA55h, читаем еще байт и делаем call.

Замечание. *Лёше привет!POST - power on self test.*

Программа инициалзации, которая идет, она узнает, что мы направили jumper'ами и все нужные значения проставит вектора на себя, САМО.

Программа. В DX порт. Потом в ecx кладем адрес + данные. Из ECX мы складываем в EAX, т.к. посылать можо только из EAX. Проверяем на -1. Если да, то перепрыгиваем печать.

Bus Device Function

Написали программу. Нужно проверять флаг multifunctional. Проверяется по флагу. Посмотреть в реестре windows - pci устройства. И, когда будет реализация талантов, можно будет сверить все.

1.11 Загрузка

Смотрели, как искать BIOS устройств. Вспоминаем байт 55 AA. E9 - короткий jmp. EA - длинный jmp. После 55 AA - идет короткий jmp E9.

Что происходит после включения компьютера?

1. Процессор тестирует себя(большая микропорграмма, прошитая в процессор).
2. Одно ядро идет на FFFF:0000. Real Mode.
3. Power On Self Test. Сначала идет тест оперативной памяти. Нужно, чтобы не было изменений в программе, которую мы пишем. Далее тест окружения (чипсет, менеджер памяти). Установка векторов. Инициализация векторов(Поиск BIOS-ов устройств и их запуск(у них есть инициализационна программа).
4. Boot Strap Loader. Int 19h. CMOS - память, поддерживаемая батарейкой. Хранит в себе информацию о Real Time Clock, настройки, пароль от BIOS . IRQ 8. Int 70. Нам нужна информация config - информация о том, откуда мы будем грузиться. IO DOC поясняет, как работать с BIOS.

В CMOS написан пароль, который считывается программой BIOS. Из CMOS можно считать. Можно найти manual о том, как считать пароль BIOS из CMOS.

Если у нас RISC архитектура, то команда умножения - длинный наобр RISC инструкций. В CISC мы дешифруем любую команду в микрокод, который уже и исполняется непосредственно процессором. Прошитая в процессор программа - набор сигналов на языке паяльника.

Предполагаем, что грузимся с HDD. С какого-то места(h7C0), первый сектор HDD диска записывается в память. BIOS в CMOS посмотрел, кто загрузочное устройство. Забрал один сектор с диска - 512 байт. Этих 512 байт хватает, чтобы считать ещё несколько байт.

MSDN AA - Microsoft открыли исходники XP, 2003 Server.

Ядро современной Windows - под 100 МБ весит ядро.

ДЗ. *На что хватит одного сектора?*

После того, как загрузилось ядро, можно указать, что грузить потом. Последовательность сектор-ядро-вся система. Между сектором и ядром - несколько инетересующих нас этапа.

Наша задача - грузиться с дискеты. Хотя грузимся с дискеты, но всё равно смотрим диск. Пишем один сектор программы. Там есть поле про размер сектора. Если это поле обнулить, то даже с дискеты грузиться не будет. Защита диск Pro.

Мы для дискеты должны не убить служебные данные и написать корректно свои данные. Нас загрузит BIOS. И нужно показать вектора и область данных BIOS(500h-600h).

Вектора - точки входа в BIOS. Узнать то, что ещё ничего не меняла

1.11.1 Реализация

Распечатали область данных (0-600h). Требование к дискете - она должна быть читаема, но должна быть и загружаема.

WinImage для создания загрузочной флешки.

Набросок структуры файловой системы. Как устроен FAT. 1 сектор - boot sector. Информация, что не загрузочная дискета, хранится на самой дискетете. Поэтому если, у нас не загрузочная флэшка, то мы распечатаем сообщение на экран с дискеты.

В boot-й сектор всегда записывается программа. Либо программа, которая печататет, что не загрузочная, либо, которая начинает загрузку.

FAT
FAT
ROOT
DATA

1 кластер = 2^n подряд идущих секторов. Сделаны кластеры для уменьшения адресации. Slack. Адрессация лежит в FAT. Записи о файлах - название, размер, дата, ссылка на первую ячейку FAT.В первой ячейкой FAT - ссылка на первую ячейку FAT.

В FAT 16 битная ячейка. Между FAT и DATA - бикекция (сколько ячеек в FAT, столько и в DATA).

ДЗ. *Какой размер FAT должен быть, чтобы натянуть на террабайтный диск.*

32-х битная спецификация по FAT. Жесткий диск делится на 4 partition. Каждый partition может быть подразбит на 4 partiotion'а. Начинаются в первом секторе и идут деревом. Каждый такой partition может содержать файловую систему или дробится на другие patition? - что-то странное.

Форматирование - нанесения файловой системы. Разметка и создание разделов. В Disk Edit нужно будет просмотреть

MBR - Master Boot Record - один сектор.

Ещё одни этам Boot Strap. MBR - тоже программа. MBR смотрит среди partition загрузочный и берет с него загрузочный. Нужно взять первый сектор и передать ему управление.

1.12 Чтение с диска

Читать будем 13 int'ом BIOS. С каждого устройства может загрузиться свой BIOS. int 13 должен поддерживаться всеми дисками, общий для всех итерфейс. Единообразную работу разрешает. Очень просто запустить процедуру форматирования. CHS - адресация по 3-м координатам 1 сектора, 512 байт. Совокупность магнитных дисков, несколько головок. Размечен - нанесена служебная информация на диск. Низкоуровневое форматирование. Не путать с форматированием - нанесением файловой системы. Цилиндры - совокупность секторов, расположенный на одном радиусе.

Отрасль вводит новый уровень абстракции. Переходит на адресацию. Сектор, это не 512 байт. Это 512 байт полезной информации. Данные о нумерации секторов записаны в отдельном секторе. Сейчас считывают целиком весь трек. Нулевого сектора не бывает - это целая метка. Даже при последовательном read приходилось снова искать метку начала. Последовательные сектора можно нумеровать через интервал - interlis. Устройство не полностью RAM. Сектора считываются один за другим. Целый лишний уровень абстрации LBA удвоил объемы жестких дисков.

Видео и жесткий диск - два основных требования пользователя.

int 13h 02h. Заполняем поля. Номер Drive - отвечает за устройство хранения. 0 - это Floppy disk. Хотим читать самый первый сектор

```
mov bx, offset buf;
mov dx, 0;
mov cx, 1;
mov ax, 201h;
int 13h;
buf db ";
```

Узнать, сколько CHS на реальном жестком диске. Какой максимальный размер LBA 8 байт? В parttition table сначала задавались?

Partition table. Как перечислять Partition table. Как задавать partition?