

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 из 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 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 в этом месте. 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 *ret*'м. Сам отладчик .COM.

Понадобится стековый 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 "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, прочесть 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 - и 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 CS, 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? Аллос (или Маллос) 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 виртуальное адресное пространство. Это поддерживается процессором. Процессор не проецирует адресное пространство на физическую память. Только после запроса адреса процессор разбирается, в какой физической памяти лежит требуемая ячейка. При обращении к неспроецированной памяти - разница в выполнении - большая.

Обработчик 14-го прерывания. Появляется в режиме protected. Выделение памяти - проецирование запрашиваемой памяти на физическую. 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 память, который мы не можем удовлетворить. Должны заниматься сбросом*

Trash - для обслуживания обращения мы часто рвем одни и те же связи, выделяя ресурсы. Алгоритмы не идеальны. Порой нет связи, которую можно порвать. Чем больше жесткой привязки, тем больше trash. Программисты решают, 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 работает 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 исполняемому процессу. Уже смотрели то, как "отрезают" память.

Говорим про системные Алло. Трехслойная архитектура Алло'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.

Вызывать 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, которую 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. (СС - 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 пройти 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/кс 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 сверху сидит BIOS. То есть 16-й вектор, который находится в 16-й ячейке памяти, указывает на BIOS. После загрузки DOS меняют 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 печатается). 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 из режима. -и режим дизассемблирования. Заметим, что вывод в little endian. Байты в обратном порядке - соглашение о размещении многобайтных величин.

Посмотрим, что в ячейке, до того, как поменяем. В одном параграфе 4 вектора, нужно пройти 5 полных параграфов и ещё 2 вектора. Посмотрели что в памяти, место где TSR. Конкретно смотрим peroges'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 аппаратное 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. Peroges просыпается по 9-му прерыванию. При alt alt alt ничего в буффер не кладем.

Оффтоп. *Можно приступить к реализации 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 минимизировать.

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

В синхронном режиме все выполняется в ритме процессора, а в асинхронной схеме обработка поступивших 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 нажатие, 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 встретится в чистом виде.

Оффтоп. *USB - поллинговое 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 происходит в любой момент. Это значит, что нужно все сохранять и восстанавливать (включая флаги). DOS это int 21h, активен, только тогда, когда мы его зовем. 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 затереть. 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 ожидание очередного int - незаметно.

1.5.4 Работы процессора

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

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

В области между завершением выполнения одной инструкции и началом декодирования другой может произойти внешнее прерывание.

Выполнение одной инструкции

1. Out of order execution. Сюда включается компиляция в микрокод. Все, что pošлем процессору компилируется в микрокод и выполняется в том порядке, который ОНИ определили.
2. Суперскалярность. Несколько конвейров для выполнения микроинструкций. Например, 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 входов, 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 один контроллер. У обоих контроллеров 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 устройств, связанных в одну сеть.

Контроллеры назывались PIC - Programming Interrupt Controller. Сейчас APIC на 24 входа.

ДЗ. Прочитать к следующему разу: *LU/io(alt + F4)* ; третий пункт, программируемый контроллер прерываний (21 страница); картинки 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 CS, а не int 3 (CD 03) - мы, захватывая 2 байта, затираем другой байт. На этот байт мог быть jmp. А в случае однобайтной инструкции CS 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 действие 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 клавиатуры 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 связывает 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'a 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 registers. 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 отпуска - 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 перешили на DirectX.

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

Попробовать добиться того, чтобы коды приходили в 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.

61 - 61 6 строк