Операционные системы

Лабораторная №1

Дизассемблирование INT 8h

Цель лабораторной работы: знакомство со средством дизассемблирования – **sourcer** и с получением дизассемблерного кода ядра операционной системы Windows на примере обработчика прерывания **Int 8h** в **virtual mode** – специальном режиме защищенного режима, который эмулирует реальный режим работы вычислительной системы на базе процессоров Intel.

Задание:

Используя sourser **(sr.exe**) получить дизассемблерный код обработчика аппаратного прерывания от системного таймера Int 8h.

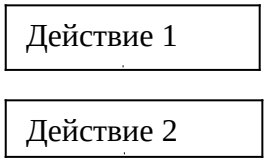
На основе полученного кода составить алгоритм работы обработчика Int 8h.

По данной лабораторной работе составляется отчет в письменном виде.

- Отчет должен содержать: полученный ассемблерный код с адресами команд и коментариями;
- Графический алгоритм работы обработчика прерывания Int 8h, структурированный и выполненный в соответствии с ГОСТ 19.701-90 ЕСПД – «Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения».
  Источник:http://www.znaytovar.ru/gost/2/GOST_1970190_ESPD_Sxemy_algori.html
  Алгоритм должен быть структурирован – теорема Бема, Якопини.

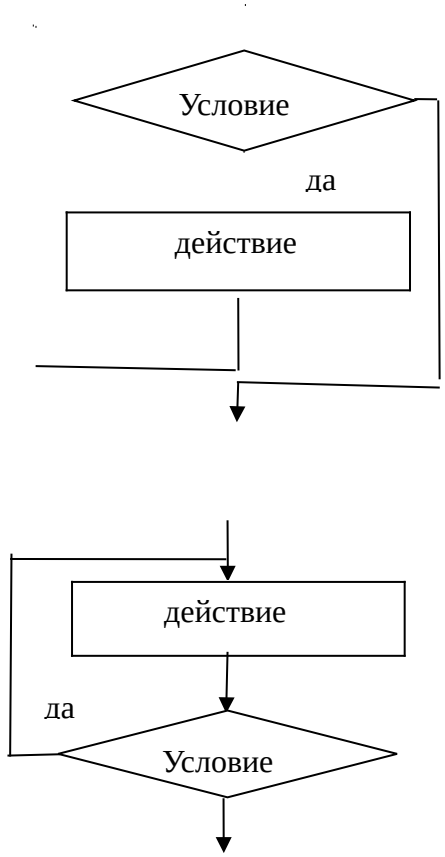
Теорема Бёма — Якопини — положение структурного программирования, согласно которому любой исполняемый алгоритм может быть преобразован к структурированному виду, то есть такому виду, когда ход его выполнения определяется только при помощи трёх структур управления: **последовательной** (англ. *sequence*), **ветвлений** (англ. selection) и повторов или **циклов** (англ. iteration).

1. Следование.



2. Ветвление.

3. Повторение – двух типов: с предусловием и постусловием



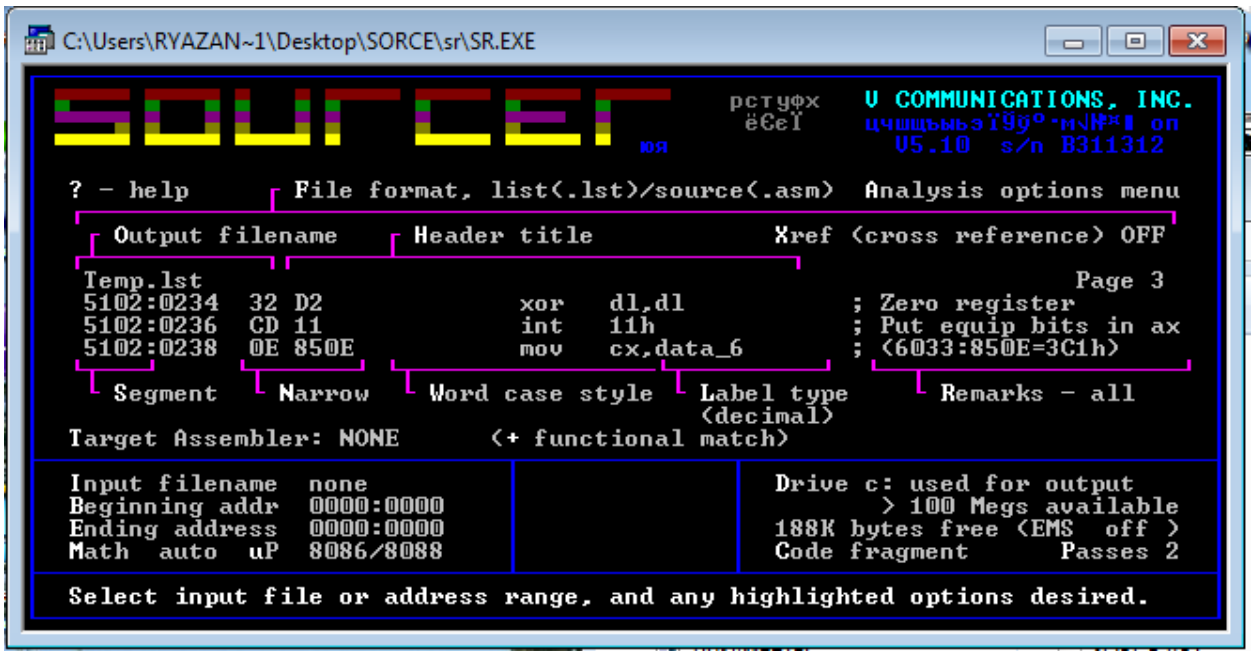
ГОСТ 19.701-90. ЕСПД. Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения

Обратите внимание, что ГОСТ называется **Схемы алгоритмов..., а не блок-схемы алгоритмов, как пишут «продвинутые» ...!!!!**

В процессорах Intel, микроконтроллерах AVR и некоторых других существуют отдельные команды для ввода-вывода — `IN` и `OUT` — и, соответственно, отдельное адресное пространство: в процессорах Intel это — адреса от 0000_{16} до $FFFF_{16}$.

Методические указания:

Сорсер не является интерактивной программой, а имеет с буквенно-цифровой интерфейс,



в котором для начала дизассемблирования надо указать начальный адрес кода - **Beginning addr** и конечный адрес - **Ending address**. Начальный адрес Вы получаете из таблицы векторов прерываний реального режима по номеру прерывания или используя 35 функцию DOS.
Конечный адрес выбирется из соображений примерного размера обработчика и затем уточняется в процессе работы. Результат работы sourcer записывается в файл **Temp.lst**. Имя файла можно изменить, нажав клавишу «o» (латинское) и в поле **Enter output file** ввести свой идентификатор. Аналогично можно изменить уровень комментариев выходного файла в окне, которое откроется при нажатии клавиши “a” (латинское) – **Analysis options menu**.

Таблица векторов прерываеий располагается в памяти начиная с нулевого адреса. Вектор прерывания это – адрес обработчика прерывания в формате сегмент:смещение. Каждый вектор занимает в таблице 4 байта: 2 байта сегмент и 2 байта смещение (так называемый, far адрес).

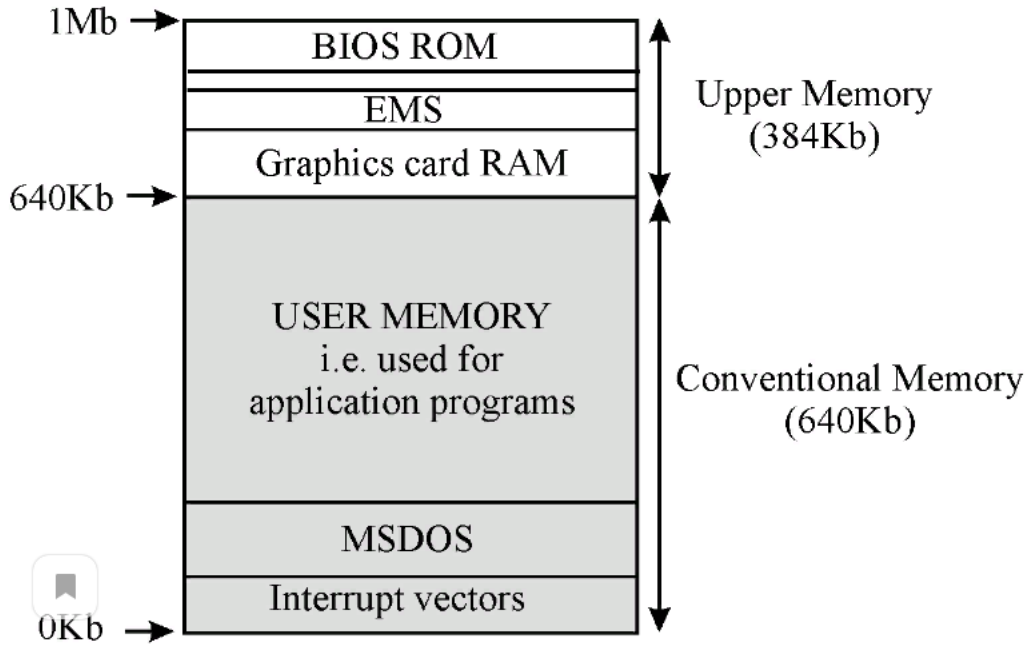


Рис. 15 базовая карта памяти системы XT под управлением MS-DOS.

Для получения адреса вектора из таблицы векторов прерываний необходимо: Вычислить смещение нужного вектора прерывания умножив номер вектора на 4 (длину far адреса в DOS). Например, вектор для 5-го прерывания (Print Screen) имеет смещение 20. Переведя 20 в шестнадцатеричное число, получим 14h, т.е. шестнадцатеричное смещение

- Используя отладчик DEBUG с помощью команд:

  DEBUG

  -D 0000:0014 L 4

  Получим содержимое четырех байтов в шестнадцетиричном виде:

  54FF:00F0

  Следует учитывать обратный порядок следования байтов:

  54FF : 00F0 , где 54ff - смещение, а 00F0 – сегмент, причем последовательность частей следующая:

  младшая : старшая и младшая : старшая части. Поменяв части местами, получим адрес – F000 : FF54

Для получения адреса обработчика можно также использовать 35 функцию ДОСа.

Код обработчика прерывания заканчивается командой iret (interrupt return). В коде эмуляции ДОСа iret надо найти по переходам и включить в листинг кода прерывания.

О функциях восьмого прерывания можно дополнительно прочитать
http://www.frolov-lib.ru/books/bsp.old/v33/ch5.htm

Аппаратное обеспечение персонального компьютера

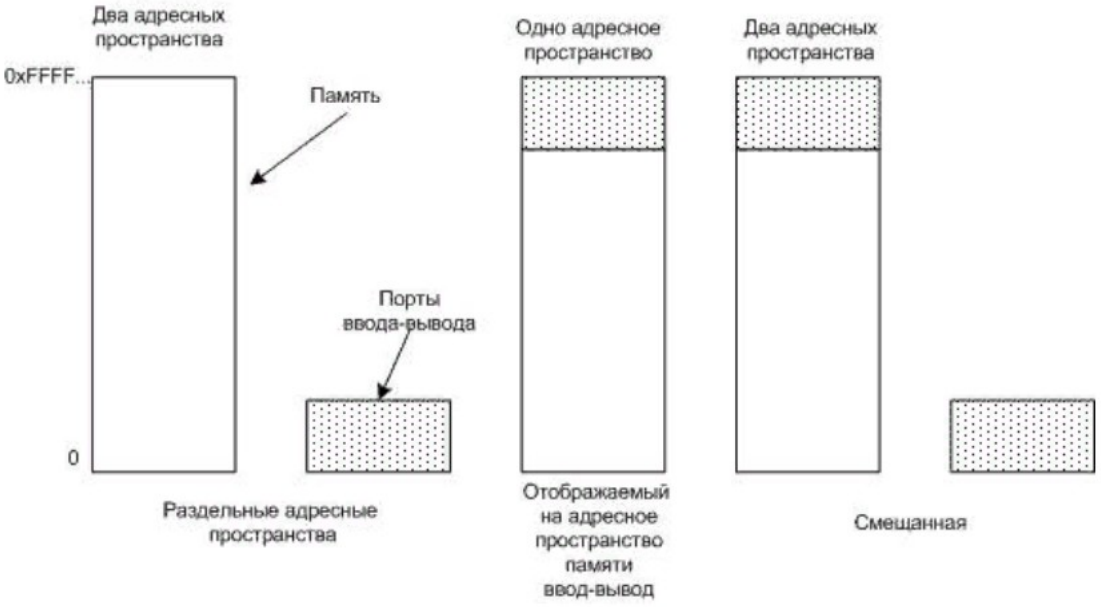
© Александр Фролов, Григорий Фролов

Том 33, М.: Диалог-МИФИ, 1997, 304 стр.

5 Системный таймер

Необходимо обратить внимание на сброс контроллера прерываний.

Дополнительно



Способы реализации доступа к управляющим регистрам и буферам

В конструкции современных персональных компьютеров используется выделенная высокоскоростная шина памяти. Эта шина предназначена для увеличения скорости обмена данными между процессором и памятью. Устройства ввода-вывода не могут «увидеть» адреса памяти, выставляемые процессором на эту шину, следовательно, они не могут реагировать на такие адреса.

Первый способ решения проблемы. Чтобы отображение регистров ввода-вывода могло работать на системах с несколькими шинами необходимо чтобы сначала все обращения к памяти посылались процессором по выделенной быстрой шине напрямую памяти (чтобы не снижать производительности). Если память не может ответить на эти запросы, процессор пытается сделать это еще раз по другим шинам.

Такое решение работоспособно, но требует дополнительного увеличения сложности аппаратуры.

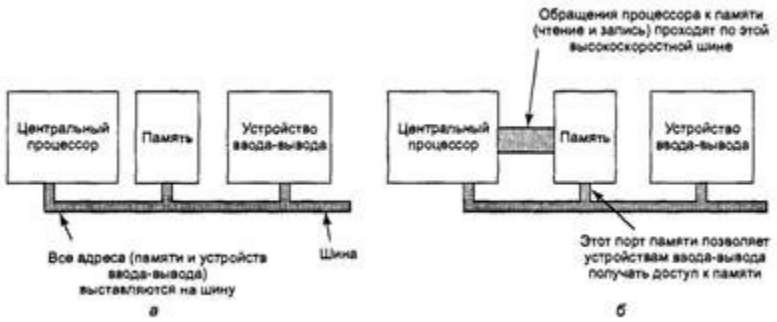


Рис. 5.2. Архитектура с одной шиной (а); архитектура памяти с двумя шинами (б)

Второй способ решения проблемы. Установка на шину памяти специального следящего устройства, передающего все адреса потенциально заинтересованным устройствам ввода-вывода. Проблема в том, что устройства ввода-вывода могут не успеть обработать эти запросы с той же скоростью, что и память.

Третий способ решения проблемы. Фильтрация адресов микросхемой моста PCI. Эта микросхема содержит регистры диапазона, заполняемые во время загрузки компьютера. Например, диапазон адресов от 640K до 1M может быть помечен как не относящийся к памяти. Все адреса, попадающие в подобный диапазон, передаются не памяти, а на шину PCI. Недостаток этой схемы состоит в необходимости принятия во время загрузки решения о том, какие адреса не являются адресами памяти.

Дополнительный материал
Зубков С.В. стр. 171-177
4.7.1 *Системный таймер*
Начиная с IBM AT, персональные компьютеры содержат два устройства для управления процессами - часы реального времени (RTC) и собственно системный таймер. Часы реального времени получают питание от аккумулятора на материнской плате и работают даже тогда, когда компьютер выключен. Это устройство можно применять для определения/установки текущих даты и времени,
установки будильника с целью выполнения каких-либо действий и для вызова прерывания IRQ8 (INT 4Ah) каждую миллисекунду. Системный таймер используется одновременно для управления контроллером прямого доступа к памяти,
для управления динамиком и как генератор импульсов, вызывающий прерывание IRQO (INT 8h) 18,2 раза в секунду. Таймер предоставляет богатые возможности для перепрограммирования на уровне портов ввода-вывода, но на уровне DOS и BIOS часы реального времени и системный таймер используются только как средство определения/установки текущего времени и организации задержек.
Функция DOS 2Ah: определить дату
Вход: AH = 2Ah
Выход: CX = год (1980-2099)
DH = месяц
DL = день
AL = день недели (0 - воскресенье, 1 - понедельник...)
Функция DOS 2Ch: определить время
Вход: AH = 2Ch
Выход: CH = час
CL = минута
DH = секунда
DL = сотая доля секунды
Эта функция использует системный таймер, поэтому время изменяется только 18,2 раза в секунду и число в DL увеличивается сразу на 5 или 6.
Функция DOS 2Bh: Установить дату
Вход: AH = 2Bh
CX - год (1980 - 2099)
DH = месяц
DL = день
Выход: AH = FFh, если введена несуществующая дата; AH = OOh, если дата установлена
Функция DOS 2Dh. Установить время
Вход: AH = 2Dh
CH = час
CL = минута
DH = секунда
DL = сотая доля секунды
Выход: AL = FFh, если введено несуществующее время, и AL = 00, если время установлено
Функции 2Bh и 2Dh устанавливают одновременно как внутренние часы DOS, которые управляются системным таймером и обновляются 18,2 раза в секунду, так и часы реального времени. BIOS позволяет управлять часами напрямую.

INT 1Ah AH = 04h: определить дату RTC
Вход: AH = 04h
Выход: CF = 0, если дата прочитана
CX = год (в формате BCD, то есть 2001h для 2001-го года)
DH = месяц (в формате BCD)
DL = день (в формате BCD)
Другие устройства
CF = 1, если часы не работают или попытка чтения пришлась на момент обновления
INT 1Ah AH = 02k. Определить время RTC
Вход: AH = 02h
Выход: CF = 0, если время прочитано
CH = час (в формате BCD)
CL = минута (в формате BCD)
DH = секунда (в формате BCD)
DL = Olh, если действует летнее время; OOh, если нет
CF = 1, если часы не работают или попытка чтения пришлась на момент обновления
INT Ш AH = 05h: установить дату RTC
Вход: AH = 05h
CX = год (в формате BCD)
DH = месяц (в формате BCD)
DL = день (в формате BCD)
INT 1Ah AH = 03h: установить время RTC
Вход: AH = 03h
CH = час (в формате BCD)
CL = минута (в формате BCD)
DH = секунда (в формате BCD)
DL = Olh, если используется летнее время, 0 — если нет
Кроме того, BIOS позволяет использовать RTC для организации будильников и задержек.
INT 1Ah AH = 06h: установить будильник
Вход: AH = 06h
CH - час (BCD)
CL = минута (BCD)
DH = секунда (BCD) .
Выход: CF = 1, если произошла ошибка (будильник уже установлен или прерывание вызвано в момент обновления часов);
CF = 0, если будильник установлен
Теперь каждые 24 часа, когда время совпадет с заданным, часы реального времени вызовут прерывание IRQ8 (INT 4Ah), которое должна обрабатывать установившая будильник программа.
Если при вызове CH = OFFh, CL = OFFh, а DH = OOh, то будильник начнет срабатывать раз в минуту.
INT 1Ah AH = 07: отменить будильник
Вход: AH - 07h
Эта функция позволяет отменить будильник, например для того, чтобы установить его на другое время.
BIOS отслеживает каждый отсчет системного таймера с помощью своего обработчика прерывания IRQO (INT 8h) и увеличивает на 1 значение 32-битного счетчика, который располагается в памяти по адресу OOOOh:046Ch, причем во время переполнения этого счетчика байт по адресу 0000h:0470h увеличивается на 1.
INT 1Ah AH = OOH: узнать значение счетчика времени
Вход: AH = OOh
Выход: CX:DX - значение счетчика
AL = байт переполнения счетчика
INT 1Ah AH = 01h: изменить значение счетчика времени
Вход: AH = Olh

CX:DX - значение счетчика
Программа может считывать значение этого счетчика в цикле (через прерывание или просто командой MOV) и организовывать задержки, например, пока счетчик не увеличится на 1. Но так как этот счетчик использует системный таймер, минимальная задержка будет равна приблизительно 55 мкс. Частоту таймера можно изменить, программируя его на уровне портов, но BIOS предоставляет для этого специальные функции.
INT 15h AH = 86h: Формирование задержки
Вход: AH = 86h
CX:DX - длительность задержки в микросекундах (миллионных долях секунды!)
Выход: AL = маска, записанная обработчиком в регистр управления прерываниями
CF = 0, если задержка выполнена
CF = 1, если таймер был занят
Если нужно запустить счетчик времени и продолжить выполнение программы, можно воспользоваться еще одной функцией.
INT 15h AH = 83h: Запуск счетчика времени
Вход: AH = 83h
AL = 0 - запустить счетчик
CX:DX - длительность задержки в микросекундах
ES:BX - адрес байта, старший бит которого по окончании работы счетчика будет установлен в 1
AL = 1 - прервать счетчик
Минимальный интервал для этих функций на большинстве систем обычно составляет около 1000 микросекунд.