Лабораторная работа №7

Задание

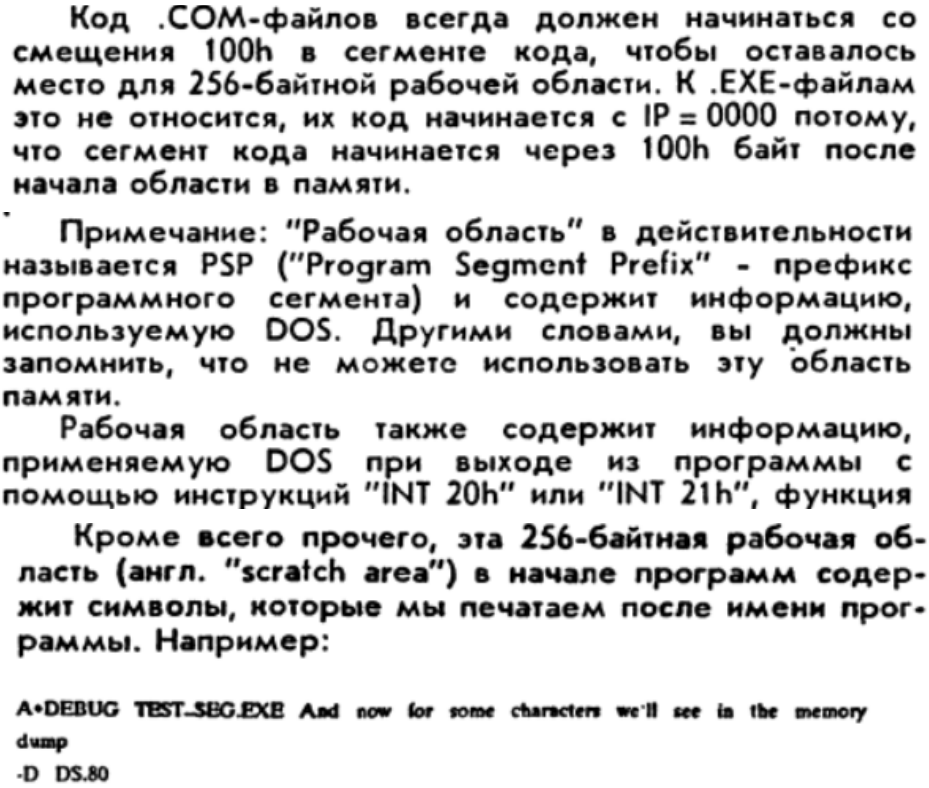
Написать резидентную программу под DOS, которая будет каждую секунду менять скорость автоповтора ввода символов в циклическом режиме, от самой медленной до самой быстрой.

Как понимать задание

Нужно написать программу, которая будет в операционку загружать обработчик прерывания (фактически, это как мы на ctrl+Z в кгшке вешаем действие возврата назад). Действием будет скорость ввода символов при долгом нажатии.

Необходимые теоретические сведения

1. Так как мы работаем в этом лр с **COM** файлом, будет полезно помнить, что в начале таких файлов всегда есть ***PSP*** – некая рабочая область, которая используется DOS и ~~куда лучше не гадить~~ к нам не относится. Поэтому важно, описывая сегмент кода, ставить сдвиг `ORG 100h`



Код .COM-файлов всегда должен начинаться со смещения 100h в сегменте кода, чтобы оставалось место для 256-байтной рабочей области. К .EXE-файлам это не относится, их код начинается с IP = 0000 потому, что сегмент кода начинается через 100h байт после начала области в памяти.

Примечание: "Рабочая область" в действительности называется PSP ("Program Segment Prefix" - префикс программного сегмента) и содержит информацию, используемую DOS. Другими словами, вы должны запомнить, что не можете использовать эту область памяти.

Рабочая область также содержит информацию, применяемую DOS при выходе из программы с помощью инструкций "INT 20h" или "INT 21h", функция

Кроме всего прочего, эта 256-байтная рабочая область (англ. "scratch area") в начале программ содержит символы, которые мы печатаем после имени программы. Например:

A>DEBUG TEST_SEG.EXE And now for some characters we'll see in the memory dump
-D DS:80

2. В чем прикол **“резидентной программы”**?

   Интересный факт: DOS – не многозадачная ОС. То есть работать можно только с одной программой (открыл текстовый редактор и пялься в него, никуда больше, без его закрытия, зайти нельзя). Собственно, чтобы это как-то обойти хитрые программисты придумали так называемые резидентные программы.

Посмотрим определение Кузнецова из лр:

> *Резидентная программа (TSR - Terminate and Stay Resident) - в операционной системе MS-DOS программа, вернувшая управление операционной системе, но оставшаяся в оперативной памяти компьютера.*

То есть фактически, резидентные программы – это такие “пчелы”, которых программист заботливо поселил в улье ОС и теперь они ему приносят счастье в виде дополнительных обработчиков каких-то действий (*ассоциация самой не нравится, но ничего лучше я не придумала, сори*). Самый банальный пример – обработчики нажатия клавиш типа Shift+Alt переключение раскладки и т.д.

Собственно об этом фраза из методички:

> *При работе с MS-DOS резидентные программы широко использовались для достижения различных целей (например, русификаторы клавиатуры, программы доступа к локальной сети, менеджеры отложенной печати).*

Как резидентная программа понимает, что пора поработать?

> *Резидентная программа активизируется каждый раз при возникновении прерывания, вектор которого эта программа изменила на адрес одной из своих процедур.*

3. Прерывания, непонятная таблица и какие-то вектора

Прерывания – своеобразный, но достаточно удобный механизм обработки асинхронных событий. Прерывание возникает в принципе при любом действии (нажатие клавиш, передвижение мыши и т.д.). Благодаря курсу кг нам проще представить себе как этот процесс работает – подобные “прерывания” мы в лабах уже обрабатывали. Собственно:

- Событие случилось
- Система запомнила набор данных о событии (какую клавишу нажали, координаты тыка мышкой и т.д.)
- Система проверяет, знакома ли она с таким событием
- Если да, то нужно выполнить некоторую процедуру, которая с ним связана.
- Возврат в тот момент, когда случилось событие

Теперь все тоже самое, но ближе к ассемблеру.

Для того чтобы проверить факт наличия описания события есть **“таблица векторов прерываний”**. Она связывает номер прерывания с адресом обработчика прерывания. Таблица занимает первый килобайт оперативной памяти (то есть, она находится в диапазоне адресов от 0000:0000 до 0000:03FFh и состоит из 256 элементов - дальних адресов обработчиков прерываний).

Из интересных номеров прерываний:

- 8h – прерывание интервального таймера, возникает 18,2 раза в секунду
- 20h-5Fh – зарезервировано/используется DOS (привет, знакомое прерывание 21h)
- 60h-67h – прерывания, зарезервированные для программ пользователя

// Как запретить прерывания? (*ну вдруг будет доп вопросом*)
Есть команда CLI. Она пишется перед блоком команд, которые обязательно должны быть выполнены вместе. Действует только на маскируемые прерывания (~~*вот здесь придется гуглить, что это*~~ это прерывания, которые можно запрещать установкой соответствующего флага).
Завершить блок нужно командой SLI, которая вернет возможность срабатывания прерываний.

4. Как заменить вектор прерывания на свою процедуру?
   Для **чтения вектора** используется функция `35h` прерывания INT 21h. Перед ее вызовом регистр AL должен содержать номер вектора в таблице. После выполнения функции в регистрах ES:BX будет искомый адрес обработчика прерывания.

   // Читаем адрес процедуры, которую вызывает 8-е прерывание
   // Фактически, загружаем в BX 0000:[AL*4], а в ES - 0000:[(AL*4)+2].
   ```
   MOV AX, 3508H
   INT 21H
   mov WORD ptr OLD_8H, BX
   ```

   Для вектора с номером, находящимся в регистре AL, функция `25h` прерывания INT 21h **устанавливает новый обработчик прерывания**. Адрес обработчика прерываний следует передать через регистры DS:DX.

   // Меняем вектор 8 прерывания
   ```
   MOV AX, 2508H
   MOV DX, OFFSET NEW_8H
   INT 21H
   ```

5. Про **порты ввода и вывода**
   – механизм взаимодействия программы, выполняемой процессором, с устройствами компьютера.

   `IN` – команда чтения данных из порта ввода `IN al, 61h`
   `OUT` – команда записи данных в порт ввода `OUT al, 61h`

6.

Парсинг программы Иры

Базовый пример хорошо рассмотрен тут

```
// Указание, что в программе один сегмент на все
.MODEL TINY

// Делаем сдвиг 100h (256 байт) для заголовка COM-файла (PSP)
CODES SEGMENT
    ASSUME CS:CODES, DS:CODES
    ORG 100H


MAIN:
    JMP INSTALL_BREAKING      // Для установки прерывания

 // Определение всяких переменных
    OLD_8H DD ?           // Сюда будем сохранять адрес старого обработчика
    FLAG DB 'E'           // Флаг, показывающий была ли установка нашего
    BUF DB 0              // Непонятная буферная хрень

    INSTALL_MSG   DB 'Breaking my_new_8h!$'
    UNINSTALL_MSG DB 'Breaking old_8h!$'

MY_NEW_8H PROC
   // Сохраняем в стек значения всех регистров
   PUSH AX
   PUSH BX
   PUSH CX
   PUSH DX
   PUSH DI
   PUSH SI

   PUSH ES
   PUSH DS

   // Команда сохраняет в стек текущее значение регистра флагов
   PUSHF

   // Вызывали старый обработчик этого прерывания (чтобы систему не ломать)
   CALL CS:OLD_8H

   // Какой-то чудо цикл до 63. Каждый раз он выполняет NEXT_STEP
   CMP BUF, 63
```

```
  JE BUF_AGAIN
  JNE CANCEL

  BUF_AGAIN:
    MOV BUF, 0
    JMP NEXT_STEP

  CANCEL:
    INC BUF

  NEXT_STEP:
```

// F3h — команда, которая отвечает за параметры режима автоповтора нажатой клавиши[1]

```
    MOV AH, 0F3h
                                          7 65 4     0
    MOV AL, 96                      // 0 11 00000
    ADD AL, BUF                     // 1 00 11110          96 + 62
```

// 60h – номер порта, в который мы хотим писать. Принимаются код команды и данные, которые хотим писать

```
    OUT 60H, AX
```

// Здесь еще есть закомментированный код

// Метка завершения прерывания (вытаскивает свое из стека и вызывает IRET)

```
  EXIT_BREAKING:
    POP DS
    POP ES
    POP SI
    POP DI
    POP DX
    POP CX
    POP BX
    POP AX

    IRET
```

// Конец процедуры-обработчика

```
MY_NEW_8H ENDP
```

---

[1] Её байт данных имеет следующее значение:
; 7 бит (старший) - всегда 0
; 5,6 биты - пауза перед началом автоповтора (250, 500, 750 или 1000 мс)
; 4-0 биты - скорость автоповтора (от 0000b (30 символов в секунду) до 11111 b - 2 символа в секунду).

```
INSTALL_BREAKING:

  // Скачали процедуру, которая обрабатывает прерывание сейчас
  MOV AX, 3508H
  INT 21H

  // Если установка уже была, то нужно наоборот вернуть все как было
  // При этом адрес старого прерывания не обновляется
  CMP ES:FLAG, 'E'
  JE UNINSTALL_BREAKING

  // Сохранили адрес старого прерывания
  MOV WORD PTR OLD_8H, BX
  // После адреса записали память                    (???)
  MOV WORD PTR OLD_8H + 2, ES

  // Установка нашего собственного прерывания
  MOV AX, 2508H
  MOV DX, OFFSET MY_NEW_8H
  INT 21H

  // Вывод сообщения о том, что установка завершена
  MOV DX, OFFSET INSTALL_MSG
  MOV AH, 9
  INT 21H

  // Прерывание 27h – завершиться, но остаться в памяти. При этом в DX кладется адрес первого байта за резидентным участком программы (что эта магическая фраза означает, для меня загадка)
  MOV DX, OFFSET INSTALL_BREAKING
  INT 27H
```

! ИТОГО: в этой метке мы узнаем, какая процедура обрабатывает прерывание сейчас. Если установка нашей уже была, то мы уходим в uninstall, иначе сохраняем старый обработчик, устанавливаем наш собственный, пишем приветственное сообщение и завершаемся, оставаясь резидентными.

```
UNINSTALL_BREAKING:
  // Сохраняем сегментные регистры, зачем, не очень ясно ?
  PUSH ES
  PUSH DS

  // Устанавливаем обратно стандартный обработчик
  // В DS:DX должен лежать его адрес (3 строчка не супер понятная)
  MOV AX, 2508H
  MOV DX, WORD PTR ES:OLD_8H
  MOV DS, WORD PTR ES:OLD_8H + 2 ?
  INT 21H

  // А теперь удаляем ¯\_(ツ)_/¯
  POP DS
  POP ES

  // Освобождаем блок памяти, начинающийся с адреса ES:0000 ?
  MOV AH, 49H
  INT 21H

  // Вывод сообщения
  MOV DX, OFFSET UNINSTALL_MSG
  MOV AH, 9H
  INT 21H

  // Завершили программу окончательно
  MOV AX, 4C00H
  INT 21H
```

! ИТОГО: эта метка нужна для того, чтобы удалить из системы наш обработчик прерывания, вернуть тот, что был, очистить всю лишнюю занимаемую память и завершить программу окончательно

```
CODES ENDS
END MAIN
```