Лабораторная работа № 11

Обработка прерываний

Цель работы: Практическое овладение навыками разработки программного кода на языке Ассемблер. Изучение основных принципов разработки пользовательских обработчиков прерываний при работе МП в реальном режиме под управлением MS DOS.

Задачи: Разработка программы, содержащей обработчик прерываний, дополняющий или заменяющий системный обработчик.

Содержание отчета:

1. Цель работы.
2. Постановка задачи (Вариант).
3. Словесный алгоритм работы программы.
4. Листинг программы.
5. Результат работы программы (если возможно).
6. Вывод.

Теоретическая часть

Прерывание (англ. interrupt) – это приостановка выполнения текущей программы в процессоре с целью выполнения другой более важной или нужной в данный момент программы или процедуры (так называемой процедуры обработки прерывания), после завершения которой, продолжается выполнение прерванной программы с точки ее прерывания.

Обработка прерываний в реальном режиме

Обработка прерываний (как внешних, так и внутренних) в реальном режиме микропроцессора производится в три этапа:

1. прекращение выполнения текущей программы;
2. переход к выполнению и выполнение программы обработки прерываний;
3. возврат управления прерванной программе.

Этап 1 - Прекращение выполнения текущей программы. Этот этап должен обеспечить временное прекращение выполнения текущей программы таким образом, чтобы потом прерванная программа продолжила свою работу так, как будто никакого прерывания не было. Любая программа, загруженная для выполнения операционной системой, занимает свое, отдельное от других программ, место в оперативной памяти. Разделяемыми между программами ресурсами являются регистры микропроцессора, в том числе регистр флагов, поэтому их содержимое нужно сохранять. Обязательными для сохранения являются регистры cs, ip и flags\eflags, поэтому они при возникновении прерывания сохраняются микропроцессором автоматически. Пара cs:ip содержит адрес команды, с которой необходимо начать выполнение после возврата из программы обслуживания прерывания, а flags\eflags — состояние флагов после выполнения последней команды прерванной программы в момент передачи управления программе обработки прерывания. Сохранение содержимого остальных регистров должно обеспечиваться программистом в начале программы обработки прерывания до их использования. Наиболее удобным местом хранения регистров является стек. В конце первого этапа микропроцессор после включения в стек регистров flags, cs и ip сбрасывает бит флага прерываний IF в регистре flags (но при этом в стек записывается предыдущее содержимое регистра flags с еще установленным IF).

Тем самым предотвращаются возможность возникновения вложенных прерываний по входу INTR и порча регистров исходной программы вследствие неконтролируемых действий со стороны программы обработки вложенного прерывания. После того как необходимые действия по сохранению контекста завершены, обработчик аппаратного прерывания может разрешить вложенные прерывания командой sti.

Этап 2 - Переход к выполнению и выполнение программы обработки прерываний. Набор действий по реализации второго этапа заключается в определении источника прерывания и вызова соответствующей программы обработки. В реальном режиме микропроцессора допускается от 0 до 255 источников прерываний. Количество источников прерываний ограничено размером таблицы векторов прерываний. Эта таблица выступает связующим звеном между источником прерывания и процедурой обработки. Данная таблица располагается в памяти, начиная с адреса 0.

Каждый элемент таблицы векторов прерываний занимает 4 байта и имеет следующую структуру:

- 1-е слово элемента таблицы — значение смещения начала процедуры обработки прерывания *(n)* от начала кодового сегмента;
- 2-е слово элемента таблицы — значение базового адреса сегмента, в котором находится процедура обработки прерывания.

Определить адрес, по которому находится вектор прерывания с номером *n,* можно следующим образом:

$$смещение\_элемента\_таблицы\_векторов\_прерываний = n * 4$$

Таким образом, полный размер таблицы векторов прерываний 4 * 256 = 1024 байт.

Теперь понятно, что на втором этапе обработки прерывания микропроцессор выполняет следующие действия:

1. По номеру источника прерывания путем умножения на 4 определяет смещение в таблице векторов прерываний.
2. Помещает первые два байта по вычисленному адресу в регистр IP.
3. Помещает вторые два байта по вычисленному адресу в регистр CS.
4. Передает управление по адресу, определяемому парой CS:IP.

Далее выполняется сама программа обработки прерывания. Она, в свою очередь, также может быть прервана, например, поступлением запроса от более приоритетного источника. В этом случае этапы 1 и 2 будут повторены для вновь поступившего запроса.

Этап 3 - возврат управления прерванной программе. Набор действий по реализации этапа 3 заключается в восстановлении контекста прерванной программы. Так же, как и на этапе 1, на данном последнем этапе есть действия, выполняемые микропроцессором автоматически, и действия, задаваемые программистом. Основная задача на этапе 3 — привести стек в состояние, в котором он был сразу после передачи управления данной процедуре. Для этого программист указывает необходимые действия по восстановлению регистров и очистке стека. Этот участок кода необходимо защитить от возможности искажения содержимого регистров (в результате появления аппаратного прерывания) с помощью команды cli. Последние команды в процедуре обработки прерывания — STI и IRET, при обработке которых микропроцессор выполняет следующие действия:

1. STI — разрешить аппаратные прерывания по входу INTR;

2. IRET — извлечь последовательно три слова из стека и поместить их, соответственно, в регистры ip, cs и flags.

В результате этапа 3 управление возвращается очередной команде прерванной программы, которая должна была выполниться, если бы прерывания не было.

Аппаратные прерывания могут быть инициированы программно командой микропроцессора int n, где n — номер аппаратного прерывания в соответствии с таблицей векторов прерываний. При этом микропроцессор также сбрасывает флаг IF, но не вырабатывает сигнал INTA.

Механизм прерываний позволяет обеспечить эффективное управление не только внешними устройствами, но и программами.

Рассмотрим разработку программного прерывания на примере.

Пример. Необходимо вывести несколько строк произвольного текста, содержащие лишь латинские буквы. Каждые 10 секунд заглавные буквы сменяются строчными и т. д.

В ходе написания программы разработаем собственный обработчик прерывания с помощью функций операционной системы DOS 25h и 35h, которые вызываются по прерыванию int 21h:

Функция 35h (INT 21h) — получить вектор прерываний
Входные данные:

- AH=35h
- AL=номер прерывания (от 00h до 0FFh)

Возвращаемые значения: ES:BX=адрес обработчика прерывания для INT AL.

Функция 25h (INT 21h) — установить вектор прерываний

Входные данные:

- AH=25h
- AL= номер прерывания (от 00h до 0FFh)
- DS:DX=адрес обработчика прерываний

Возвращаемые значения: Функция ничего не возвращает.

Текст программы

```
.model small
.stack 100h
.486
.data
 old_handler_offset dw ?
 old_handler_seg dw ?
 printed db '0'
 counter dw 0ffffh
 prev_counter dw 0
 period dw 182 ; 10 секунд / 55 мс = 181.81
 color db 0fh
 exit_flag db 0
 output_str db "Jingle bells, jingle bells Jingle all the way ", '$'
```

```
  .code
  inter_handle proc far
       pusha
       mov ax, data
       mov ds, ax
; буфер ввода
       mov ax, 0040h
       mov es, ax
       mov bx, es:[001ah] ; начало
       mov ax, es:[001ch] ; конец
       cmp ax, bx
       je no_input ; buffer length = 0
            mov al, es:[bx]
            mov es:[001ch], bx
            cmp al, 27 ; ESC
            jne skip_exit
               mov exit_flag, 1
            skip_exit:
            cmp al, '0'
            jne
            skip_exit2
               mov exit_flag, 1
            skip_exit2:
       no_input:
       mov ax, counter
       sub ax, prev_counter
       cmp ax, period
       ja print ; если больше периода - обновляем
       jmp skip_print
       print:
            mov ax, counter
            mov prev_counter, ax
; записываем адрес графического буфера в es
            mov ax, 0B800h
            mov es, ax
            mov si, 0
            mov cx, 25 ; количество строк - 25
            swap_loop:
               push cx
```

```
                ; для каждого символа в строке
                mov cx, 80
                line_loop:
                ; меняем регистр es:[si]
                 mov ax, es:[si]
                 cmp al, 'a'
                 jb skip_lowercase cmp al, 'z'
                 ja skip_lowercase
                   sub al, ' '
                   jmp skip_uppercase
                skip_lowercase:
                cmp al, 'A'
                jb skip_uppercase
                cmp al, 'Z'
                ja skip_uppercase
                    add al, ' '
                skip_uppercase:
                mov es:[si], ax
                ; переход к следующему символу
                add si, 2
                loop line_loop
            pop cx
          loop swap_loop
     skip_print:
     inc counter
     popa
     iret
inter_handle endp
; Подпрограмма очистки экрана
 cls proc near
     push cx
     push ax
     push si
     xor si, si
     mov ah, 7
     mov al, ' '
     mov cx, 2000
     CL1:
          mov es:[si], ax
          inc si
          inc si
```

```
            loop CL1
        pop   si
        pop   ax
        pop   cx
        ret
  cls endp

; Основная программа
  start:
        mov ax, @data
        mov ds, ax
; записываем адрес графического буфера
        mov ax, 0B800h
        mov es, ax
; очищаем экран
        call cls
; сохраняем старый вектор прерывания
        mov ah, 35h
        mov al, 1Ch
        int 21h
        mov old_handler_offset, bx
        mov old_handler_seg, es
;устанавливаем свой вектор прерывания на функцию
        ; inter_handle
        push ds
          mov dx, offset inter_handle
          mov ax, seg inter_handle
          mov ds, ax
          mov  ah,  25h
          mov  al,  1Ch
          int 21h
        pop ds
  ; заполняем экран
        mov ah, 09h
        mov dx, offset output_str
        mov  cx,  20
        fill_loop:
             int 21h
             loop    fill_loop
        main_loop:
             nop ;задержка ≈2 мкс
             nop
             nop
             cmp exit_flag, 1
```

```
        je break_main_loop
        jmp main_loop
    break_main_loop:
   ; возвращаем старый вектор прерывания push ds
    mov dx, old_handler_offset
    mov ax, old_handler_seg
    mov ds, ax
    mov ah, 25h
    mov al, 1Ch
    int 21h
    pop ds
    mov ax,0600H
    mov bh, 0fh
    mov cx,0000
    mov dx, 184FH
    int 10H
   ;вызов сервиса DOS для завершения программы
    mov ax, 4c00h
    mov al, 0
    int 21h
end start
```

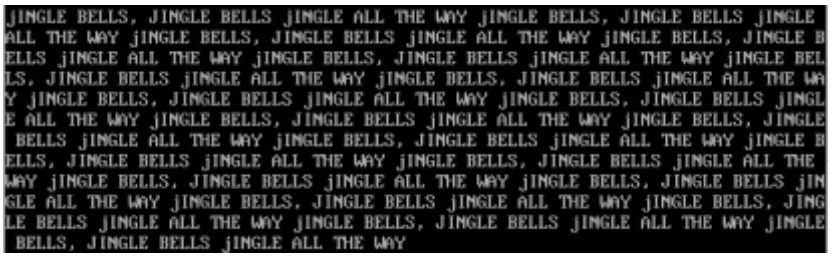

Рис. 1. Результат программы Шаг 1.

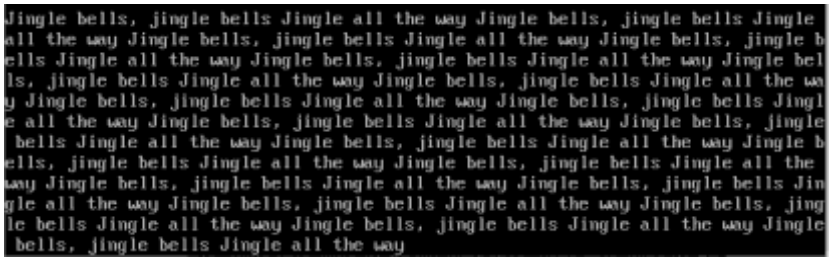

Рис. 2. Результат программы Шаг 2.

ВАРИАНТЫ

Во всех вариантах задания завершение программы осуществляется при вводе цифры 0.

1. Составить программу «будильник», выдающую на экран строку «Время истекло!» после запуска по истечении некоторого промежутка времени (в секундах). Время задержки определить самостоятельно. Используйте прерывание 08H или 1CH и 10H. Завершение программы осуществляется при вводе цифры 0.
2. Выводить последовательно цифры от 0 до 9 в одно место экрана. При нажатии на клавиатуре клавиш “1”, “2”, “3”, “4” меняется темп вывода (для анализа нажатия клавиши использовать вектор 1Ch). Значение задержки между выводом очередного символа определять следующим способом: введённую цифру умножить на 29, это и будет число повторений цикла задержки. Завершение программы осуществляется при вводе цифры 0.
3. Составить программу перемещения произвольного символа из левого верхнего угла экрана в правый нижний угол по истечении промежутка времени, время выбрать самостоятельно. При нажатии на клавиатуре клавиш “1”, “2”, “3” меняется темп вывода (для анализа нажатия клавиши использовать вектор 1Ch). Значение задержки между выводом очередного символа определять следующим способом: введённую цифру умножить на 29, это и будет число повторений цикла задержки. Завершение программы осуществляется при вводе цифры 0.
4. Очистить экран. Вывести несколько строк произвольного текста (атрибут 14). Перехватив прерывание печати экрана Print Screen (Int 5h), менять атрибуты всех строк экрана циклически от 1 до 15. Каждое нажатие клавиши Print Screen вызывает изменение атрибута. Завершение программы осуществляется при вводе цифры 0.
5. Выводить ежесекундно системное время “часы:минуты:секунды”. При каждом выводе менять положение вывода случайным образом, циклически меняя атрибуты символов. Завершение программы осуществляется при вводе цифры 0.
6. Выводить в одно место экрана поочерёдно код пробела и код символа “$”. В процессе вывода на каждом шаге менять атрибут цвета символа. При нажатии на клавиатуре клавиш “5”, “6”, “7”, “8” меняется темп вывода (для анализа нажатия клавиши использовать вектор 1Ch). Задержка между выводом каждого символа определяется нажатием цифровой клавиши, следующим способом: введённую цифру умножить на 29, это и будет число повторений цикла задержки. Завершение программы осуществляется при вводе цифры 0.
7. Выводить в текущее положение курсора символ #. Следующий символ # выводить в позицию выше, ниже, левее или правее текущего символа, в зависимости от нажатия клавиш “8”, “2”, “4”, “6” на цифровой клавиатуре. При смене позиции символа меняется атрибут. Вывод осуществлять непрерывно с некоторой задержкой. Для анализа нажатия клавиши использовать вектор 1Ch. Завершение программы осуществляется при вводе цифры 0.
8. В программе имеются два циклических счётчика, считающих от 0 до 23 и от 0 до 79. Их значение определяет соответственно строку и столбец для вывода символа на экран. При нажатии клавиши “A” на экран выводится символ % в положение, определяемое состоянием счётчиков на момент вывода. Для анализа нажатия клавиши использовать вектор 1Ch. Завершение программы осуществляется при вводе цифры 0.
9. В программе имеется циклический счётчик, считающий от 0 до 9. При нажатии любой клавиши содержимое счётчика преобразуется в ASCII код и выводится в

определённое место экрана (атрибут меняется при каждом выводе), после чего счётчик продолжает считать. Для анализа нажатия клавиши использовать вектор 1Ch.

10. Очистить экран. Вывести несколько строк произвольного текста (атрибут 14) в середине экрана. Перехватить прерывание 1Ch, по нажатию клавиши ‘1’ осуществить горизонтальный скроллинг всего экрана влево на один столбец, при нажатии клавиши ‘2’ скроллинг вправо на один столбец. Завершение программы осуществляется при вводе цифры 0.
11. В программе имеются два циклических счётчика, считающих от 0 до 23 и от 0 до 79. Их значение определяет соответственно строку и столбец для вывода на экран. последовательность степеней числа 2. Завершение программы осуществляется при вводе цифры 0 или когда числа выйдут за пределы 16-разрядной сетки.
12. Посчитать за какое время процессор выполнить 1 000 000 команд mov DI, SI; add DI, SI; mul SI. Для подсчёта времени использовать вектор 1Ch. Вывести на экран, в место указанное курсором мыши, преобразованное в ASCII коды число тиков таймера, затраченное на операцию.
13. Дан массив A из 10 однобайтных чисел. Перехватив прерывание от таймера вывести из массива A на экран все числа, большие 05h и меньшие 20h. Повторять вывод каждые 5 секунд, сдвигаясь на две строки вниз. Завершение программы осуществляется при вводе цифры 0.
14. В программе имеется циклический счётчик, считающий от 00h до FFh. Его значение преобразуется в ASCII код и выводится в левом верхнем углу экрана через 18 тиков таймера. При нажатии клавиши ‘2’ время вывода уменьшается вдвое, а при повторном нажатии время вывода увеличивается в два раза. Для анализа нажатия клавиши и подсчёта числа тиков таймера использовать вектор. Завершение программы осуществляется при вводе цифры 0.
15. Очистить экран. Вывести несколько строк произвольного текста. Перехватить прерывание экрана (Int 5h). Первый вызов этого прерывания располагает строки вертикально, следующий горизонтально и т.д. При нажатии на клавиатуре клавиш “2”, “4”, “6”, “8” меняется темп вывода (для анализа нажатия клавиши использовать вектор 1Ch). Значение задержки между выводом очередного символа определять следующим способом: введённую цифру умножить на 29, это и будет число повторений цикла задержки. Завершение программы осуществляется при вводе цифры 0.
16. В массиве из 16 ячеек памяти располагаются шестнадцатеричные числа от 00 до 00F. В массиве есть только одно число, которое повторяется несколько раз. При первом нажатии на клавишу “1” выявить какое это число, при повторном нажатии сколько раз оно повторяется. Результат разместить в центре экрана. Завершение программы осуществляется при вводе цифры 0.
17. Очистить экран. Заполнить его произвольной информацией. Перехватить прерывание экрана (Int 5h). Первый вызов этого прерывания переносит строчки верхней половины экрана на место нижних, а нижние на место верхних. Следующий вызов прерывания снова меняет их местами и т.д. Завершение программы осуществляется при вводе цифры 0.
18. В программе имеется циклический счётчик, считающий от 00h до FFh. Его значение преобразуется в ASCII код и выводится в произвольном месте экрана через 18 тиков таймера, каждый раз меняя значение атрибута Завершение программы осуществляется при вводе цифры 0.

19. Очистить экран. Вывести несколько строк произвольного текста (атрибут 14). Перехватить прерывание 1Ch, по нажатию клавиши на одну из клавиш 1...9 осуществить горизонтальный скроллинг всего экрана влево на соответствующее число столбцов. Завершение программы осуществляется при вводе цифры 0.
20. В программе имеется циклический счётчик, считающий от 00h до FFh. При нажатии любой клавиши содержимое счётчика преобразуется в ASCII код и выводится в определённое место экрана (атрибут меняется при каждом выводе), после чего счётчик продолжает считать. Завершение программу можно досрочно, нажав на клавишу 0.
21. Очистить экран. Вывести несколько строк произвольного текста. Перехватить прерывание экрана (Int 5h). Первый вызов этого прерывания располагает строки вертикально, следующий горизонтально и т.д. При нажатии на клавиатуре клавиши “8” меняется цвет выводимого текста. Завершение программы осуществляется при вводе цифры 0.
22. Выводить указанное курсором место экрана символ до тех пор, пока не будет выбрано новое место. При смене места вывода символ и атрибут символа. Для анализа нажатия клавиши использовать вектор 1Ch.
23. Написать программу получения плавного перехода тонов – «пожарная сирена». Генерацию звука производить путём программирования таймера.
24. Написать программу получения плавного перехода тонов – «падающий метеорит». Генерацию звука производить путём управления динамиком непосредственно процессором (использование таймера запрещено).
25. Написать программу проигрывания мелодии. Таблица номеров частот и длительностей нот: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22. Таблица частот: 1709, 1809, 2031, 2280, 1521, 1709, 1809, 2031, 2280, 1521, 1709, 1353, 1709, 1809,1521,1809, 2031, 1809, 1709, 2031, 2280, 1139. Таблица задержек: 4, 4 ,4, 4, 5, 4, 4, 4, 4, 5, 4, 5, 4, 4, 5, 4, 4, 4, 4, 4, 5, 5. Генерацию звука производить путём программирования таймера.
26. Написать программу проигрывания мелодии. Таблица номеров частот и длительностей нот: 6, 5, 4, 3, 1, 1, 6, 5, 4, 3, 1, 1, 6, 2, 2, 6, 5, 1, 1, 5, 4, 5, 6, 4, 3, 3, 6, 2, 2, 6, 5, 1, 1, 5, 4, 5, 6, 4, 3, 3. Таблица частот: 1140, 1015, 1705, 1520, 1355, 1205. Таблица задержек: 8, 8, 8, 8, 4, 4, 8, 8, 8, 8, 4, 4, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 4, 4, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 4, 4. Генерацию звука производить путём программирования таймера
27. Составить программу выдачи краткого комментария, если курсор находится в начале или конце ключевого слова. Порядок расположения слова (слева или справа от курсора) определяется командной строкой. Используйте прерывание 10H.
28. Составить программу выдачи скан-кода символа, указанного курсором из множества символов {0…9}. Используйте прерывание 10H.