МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Организация ЭВМ и систем»

Тема: Написание собственного прерывания и работа со строками

Студент гр. 2384 _____ Кузьминых Е.М.

Преподаватель _____ Морозов С.М.

Санкт-Петербург

2023

Цель работы.

Изучение того как работают прерывания, написание своего собственного прерывания и освоение работы со строками в программах на языке Ассемблера процессора Intel X86.

Основные теоретические положения.

1. Краткие сведения о прерываниях.

Прерывание – это процесс вызова процедур для выполнения некоторой задачи, обычно связанной с обслуживанием некоторых устройств (обработка сигнала таймера, нажатия клавиши и т.д.). Когда возникает прерывание, процессор прекращает выполнение текущей программы (если её приоритет ниже) и запоминает в стеке вместе с регистром флагов адрес возврата (CS:IP) – места, с которого будет продолжена прерванная программа. Затем в CS:IP загружается адрес программы обработки прерывания и ей передаётся управление.

Адреса 256 программ обработки прерываний, так называемые векторы прерывания, имеют длину по 4 байта (в первых двух хранится значение IP, во вторых – CS) и хранятся в младших 1024 байтах памяти.

Программа обработки прерывания должна заканчиваться инструкцией IRET (возврат из прерывания), по которой из стека восстанавливается адрес возврата и регистр флагов.

Программа обработки прерывания – это отдельная процедура, имеющая структуру:

```
SUBR_INT PROC FAR
 PUSH AX ; сохранение изменяемых регистров

 <действия по обработке прерывания>

 MOV AL, 20H
 OUT 20H, AL
 POP AX ; восстановление регистров
 IRET
SUBR_INT ENDP
```

Две последние строки перед восстановлением регистров необходимы

для разрешения обработки прерываний с более низкими уровнями, чем только что обработанное. Эта добавка не нужна для тех векторов прерываний, которые являются расширениями существующих прерываний, таким как прерывание 1CH, которое добавляет код к прерыванию времени суток.

Программа, использующая новые программы обработки прерываний, при своём завершении должна восстанавливать оригинальные векторы прерываний. Функция 35 прерывания 21H возвращает текущее значение вектора прерывания, помещая значение сегмента в ES, а смещение в BX. В этом случае программа должна содержать следующие инструкции:

```
    ; -- в сегменте данных
    KEEP_IP DW 0 ; для хранения смещения
    KEEP_CS DW 0 ; и сегмента прерывания
    KEEP_OLD_VEC EQU KEEP_IP ; опционально, для удобства перехода на старое
тело

    ; -- в начале программы
    MOV AH, 35H ; функция получения вектора
    MOV AL, 1CH ; номер вектора
    INT 21H
    MOV KEEP_IP, BX ; запоминание смещения
    MOV KEEP_CS, ES ; и сегмента
```

Для задания адреса собственного прерывания с заданным номером в таблицу векторов прерываний используется функция 25H прерывания 21H, которая устанавливает вектор прерывания на указанный адрес нового обработчика.

```
    PUSH DS
    MOV DX, OFFSET ROUT ; смещение для процедуры в DX
    MOV AX, SEG ROUT ; сегмент процедуры
    MOV DS, AX ; помещаем в DS
    MOV AH, 25H ; функция установки вектора
    MOV AL, 60H ; номер вектора
    INT 21H ; меняем прерывание
    POP DS
```

В конце программы восстанавливается старый вектор прерывания:

```
    ; CLI
    PUSH DS
    MOV DX, KEEP_IP
```

```
MOV AX, KEEP_CS
MOV DS, AX
MOV AH, 25H
MOV AL, 1CH
INT 21H ; восстанавливаем вектор
POP DS
; STI
```

При использовании функции 25H прерывания 21H DOS знает, что вы делаете, и гарантирует, что в момент записи прерывания будут заблокированы. Поэтому вызывать команды CLI и STI не нужно. Но они понадобятся в Ыслучае ручного изменения вектора прерывания (т.е. без вызова INT 21H), чтобы не допустить возможного возникновения ужасной ситуации, когда сегмент был переопределён, а смещение осталось старым (или наоборот).

Прерывания бывают аппаратные (вызываемые в результате сигналов от оборудования) и программные (вызываемые в коде). В лабораторной работе предлагаются к замене вектора́ следующих прерываний:

– 1CH и 08H – можно рассматривать их как аппаратные прерывания, генерируемые системным таймером; генерируются автоматически 18,2 раза в сек.;

– 09H – аппаратное прерывание, возникающее в результате нажатия клавиш клавиатуры;

– 60H – пользовательское программное прерывание;

– 16H – программное прерывание для ожидания ввода символа с клавиатуры;

– 21H – программное прерывание для вызова сервисов DOS.

Заменённое тело 9h следует завершать не выходом из прерывания (iret), а переходом к выполнению старого тела 9h (использовать команду jmp dword ptr), иначе обработка сигналов клавиатуры будет нарушена. То же самое касается прерываний 16h и 21h.

2. Краткие сведения о командах обработки строк.

Для обработки строковых данных ассемблер имеет пять групп команд обработки строк:

— MOVS — переслать один байт или одно слово из одной области памяти в другую;

— LODS — загрузить из памяти один байт в регистр AL или одно слово в регистр AX;

— STOS — записать содержимое регистра AL или AX в память;

— CMPS — сравнить содержимое двух областей памяти, размером в один байт или в одно слово;

Каждая команда имеет модификации, указывающие размер операнда: байт (B), слово (W), двойное слово (D). Например: MOVSB, MOVSW, MOVSD.

Эти команды предполагают, что регистры DI и SI содержат относительные адреса, указывающие на необходимые области памяти (для загрузки можно использовать команду LEA). Регистр SI обычно связан с регистром сегмента данных – DS:SI. Регистр DI всегда связан с регистром дополнительного сегмента – ES:DI. Следовательно, команды MOVS, STOS, CMPS требуют инициализации регистра ES (обычно адресом в регистре DS).

Префиксы REP/REPE/REPZ/REPNE/REPNZ позволяют этим командам обрабатывать строки любой длины.

В следующем примере выполняется пересылка 20 байт из STRING1 в STRING2. Пусть оба регистра DS и ES уже инициализированы адресом сегмента данных:

```
STRING1 DB 20 DUP('*')
STRING2 DB 20 DUP(' ') ...
CLD ;Сброс флага DF для направления слева направо
MOV CX,20 ;Счётчик на 20 байт
LEA DI,STRING2 ;Адрес области "куда"
LEA SI,STRING1 ;Адрес области "откуда"
REP MOVSB ;Переслать данные
```

3. Необходимые вспомогательные функции.

3.1. Ввод строки:

```
; -- в сегменте данных
EOFLine  EQU  '$' ;Определение символьной константы "Конец строки"

    STR1HEAD DB 50h,0 ; заголовок строки: можно ввести максимум 50h=80
символов. В следующем байте вместо 0 появится фактическое количество введённых
символов
```

```
    STR1 DB 80 DUP('*'),0AH, 0DH, EOFLine ; Буфер памяти для введённых символов
плюс байты для корректного завершения вывода

    ; -- в сегменте кода
    StringRead PROC FAR
     mov AH,0ah ; функция ввода строки
     push dx  ; смещение заголовка строки...
     int 21h                                  ; вызов функции DOS ввода строки
     pop bp ; ...поместить в bp
     xor bx,bx
     mov bl,ds:[bp+1] ; теперь в bx количество введённых символов
     add bx,bp ; теперь bx указывает на конечный введённый символ
     add bx,2 ; теперь bx указывает на байт, следующий за финальным 0dh
     mov word ptr [bx+1],240ah ;добавить в конец 0ah и '$'
     ret
    StringRead ENDP

    ...
    mov dx, OFFSET STR1HEAD ; вызов функции ввода строки
    call StringRead
```

3.2. Задержка во времени:

```
mov cx,0eh; 14 * 65535 мкс задержки
mov dx,0ffffh; ещё 65535 мкс задержки
mov ah,86h; функция "ждать"
int 15h; вызов функции ожидания
```

Задание

Состоит из двух основных задач:

1) Реализация сценария работы с прерываниями в соответствии с вариантом.

2) Реализация преобразования строки с использованием команд работы со строками в соответствии с вариантом.

Вариант 14. Шифр 14м.

- Действия основной программы: замена 9h, замена 60h, ожидание (5 секунд), вызов 60(h)
- Действия 9h: отметить, что был вызван.
- Действия 60h: выполнить работу, если ранее был вызван 9h.
- Вариант преобразования строки: м) В выходную строку передаются сначала все цифры из входной строки, а потом все остальные символы. Порядок цифр сохраняется, а порядок остальных символов инвертируется.

Замечания:

1) В сценариях опущены:

– Вывод приветственного сообщения и ввод строки пользователем. Это 1-ый пункт основной программы каждого сценария (поэтому основная программа во всех сценариях начинается с пункта 2). Слова «Ожидание ввода строки», которые встречаются в таблице сценариев, – это всего лишь организация задержки с ожиданием нажатия клавиш, для ввода данных не используется.

– Действия по восстановлению изменённых прерываний, если они не требуются по сценарию специально. После завершения программы все изменённые прерывания всегда должны быть восстановлены (независимо от сценария).

2) «вып. работу» означает, что надо на основе введённой строки (п. 1 основной программы) создать модифицированную строку и вывести её на экран. При выполнении преобразования нельзя портить исходную строку, результат преобразования должен записываться в выходную строку.

3) Перед запуском ожидания нажатия клавиши («ввод строки» в таблице) вывести сообщение об этом.

4) Перед заменой 9h следует сделать небольшую задержку (см. 3.2 «Задержка во времени»), чтобы предшествующая активность пользователя была обработана до того, как 9h будет изменён;

5) Для исключения возможного взаимного влияния системных и пользовательских прерываний рекомендуется отвести в программе под стек не менее 1Кбайт.

Замечания:

1) При выполнении преобразования обязательно использовать команды работы со строками.

2) Завершающие символы (0ah, 0dh, «$») рассматривать именно как завершающие, т.е. не подвергать преобразованиям; закончить ими выходную строку.

Выполнение работы.

Сегмент стека: запрашивается 1 кБ памяти для стека. Команда DW (define word) резервирует 512 слов (где каждое слово в данном контексте равно 2-м байтам). Данная область памяти будет использоваться для временного хранения данных.

Сегмент данных: определяется набор переменных различных типов. В частности, используются байтовая переменная MAX_LEN (максимальная длина вводимой строки) и другие переменные для хранения флагов и счетчиков.

Сегмент кода: указывает, что регистры CS (регистр сегмента кода), DS (регистр сегмента данных) и SS (регистр сегмента стека) ссылается на определенные сегменты памяти.

Процедура Print: эта процедура принимает строку и выводит её на экран с использованием прерывания DOS 21h с AH установленным на значение 9, которое обозначает функциональную команду для вывода строки.

Процедура ReadString: Сохраняем регистры. Мы вставляем регистры ax, bp и bx в стек. Это делается для сохранения их текущего состояния, поскольку всех этих регистров собираются использовать в процедуре и в конце процедуры они будут восстановлены. Затем в регистре ah ставим команду 0ah (считывание строки от пользователя и сохранение её в буфере), а в регистр dx ставим указатель на начало буфера, где будет сохраняться считанный текст. Прерывание int 21h используется здесь для вызова этой команды DOS. Регистр bx набором команд обнуляется, а затем используется для хранения размера введенной строки (хранится в следующем байте после начала строки). Добавляя bp (адрес начала строки) к bx и двойку, мы указываем bx на конец строки. Добавляем символы конца строки и перевода строки. 0ah, 24h (символ перевода строки и символ конца строки) добавляются в конец считанной строки.

Восстанавливаем регистры, используется ret для возвращения управления вызывающей программе.

Процедура RestoreInterruptions: В начале процессы сохраняются значения регистров AX, DS и DX, чтобы их текущие значения не были случайно изменены в процессе выполнения этого кода. Далее код проверяет значение переменной overwritten. Если она равна нулю, то этот код уже был выполнен ранее и процесс восстановления прерываний не требуется. В случае, если переменная overwritten не равна нулю, код продолжает с тем, чтобы восстановить прерывания. Процедура восстановления выполняется отдельно для каждого прерывания. Сначала восстанавливается прерывание 60h, затем прерывание 9h. Сохранение и восстановление DS используются для защиты и восстановления значения этих регистров, поскольку они изменяются в ходе выполнения кода. В концовке, после восстановления прерываний, переменная overwritten устанавливается обратно в значении 0. Наконец, происходит восстановление изначального состояния регистров DX, DS и AX, и функция завершает свою работу.

Процедура ChangeInterruptions: Сначала она сохраняет регистры ax, bx, es. Затем происходит сравнение переменной overwritten со значением 1. Если она равна 1, это означает, что прерывания уже были перезаписаны и производится прыжок на метку already_changed. Если это не так, функция продолжает свою работу. Функция сначала выполняет копирование оригинального вектора прерывания 9h. Полученные значения сохраняются в KEEP_IP_9h и KEEP_CS_9h. Затем она задает новый обработчик прерываний для 9h — IRS_9h. Это делается путем загрузки смещения и сегмента IRS_9h в регистры dx и ax соответственно. Точно такой же процесс повторяется для вектора прерывания 60h. В конце функция устанавливает значение overwritten в 1. На метке already_changed происходит восстановление оригинальных значений регистров, и функция завершает свою работу.

Процедуры IRS_60h и IRS_9h: IRS_60h - Это прерывание сохраняет значения регистров AX, SI, DI и DX (чтобы после выполнения прерывания вернуть их в оригинальное состояние) и проверяет было ли вызвано прерывание IRS_9h (переменная called). Если прерывание IRS_9h не было вызвано, прерывание IRS_60h сразу же завершает свою работу и восстанавливает регистры. Если IRS_9h было вызвано, то следующий код переносит последовательность цифр из массива original в массив processed, пропуская все символы, не попадающие в диапазон '0'-'9'. После этого, этот же набор чисел записывается обратно в массив processed, но в обратном порядке. В случае если символ не является числом, он пропускается.
После успешного завершения этих действий, в processed записывается символ новой строки (символ '\n') и запускается подпрограмма Print, которая выводит данные из processed. После этого прерывание завершается, восстанавливает регистры и отправляет команду End Of Interrupt (EOI) контроллеру прерываний для разрешения обработки других прерываний. IRS_9h - Это прерывание проставляет флаг called в 1, указывающий, что оно было вызвано. Затем оно прыгает к оригинальному обработчику прерывания 9h, хранящемуся в KEEP_IP_9h.

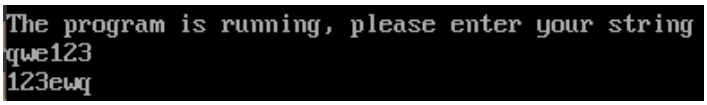
Процедура Main: это основная процедура программы, которая контролирует процесс выполнения программы. Она вызывает все остальные процедуры в определенной последовательности для реализации требуемого функционала.

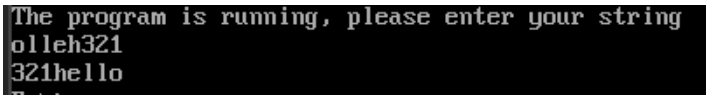
Тестирование.

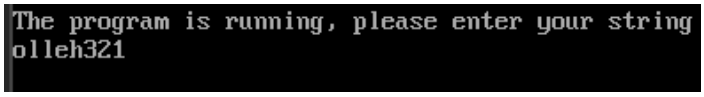
Табл. 1

Входные данные	Выходные данные	Комментарий
qwe123 (клавиша нажата)	123ewq	Ответ верный
olleh321	321hello	Ответ верный

(клавиша нажата)		
olleh321 (клавиша не нажата)	Пустой ввод	Ответ верный, прерывание не запустилось







Вывод.

По итогам лабораторной работы были изучены основы создания своих прерываний и обработки строки. В результате была написана программа, переопределяющая два прерывания – 9h и 60h, считывающая строку и изменяющая ее, инвертировав символы – не цифры, а также выводя цифры в начало строки.

ПРИЛОЖЕНИЕ

Код программы lb4.asm

```
AStack SEGMENT STACK
      DW 512 DUP(?)                ; выделили 1кб стека
AStack ENDS

DATA SEGMENT
      MAX_LEN     equ 64h          ; макс длина строки - 100 символов
```

```
      header            DB MAX_LEN, 0                                    ;
заголовок для входной строки
      original    DB MAX_LEN DUP('?'), 0dh, 0ah, '$'  ; буфер входной строки
      processed   DB MAX_LEN DUP('?'), 0dh, 0ah, '$'  ; буфер строки для
обработки
      greeting    DB 'The program is running, please enter your string', 0dh,
0ah, '$'
      waiting     DB 'Waiting for input...', 0dh, 0ah, '$'
      counter     DB 0        ; счетчик для вычисления вызовов прерывания 1ch
      called            DB 0        ; флаг, который показывает, был ли вызван 9h
      overwritten DB 0        ; флаг, который показывает, перезаписаны ли 9h и
60h
      KEEP_IP_60h DW 0        ; переменные для хранения ip, cs, ip, cs 60h
      KEEP_CS_60h DW 0
      KEEP_IP_9h DW 0
      KEEP_CS_9h DW 0
DATA ENDS

CODE SEGMENT
      ASSUME CS:CODE, DS:DATA, SS:AStack

Print PROC
      ;сохранил регистры
      push ax

      mov ah, 9h        ; в 9h строка, которая будет печататься в терминал
      int 21h                 ; печатаю строку

      pop ax
      ret
Print ENDP

ReadString PROC
      ;сохранил регистры
      push ax
      push bp
      push bx

      mov ah, 0ah
      push dx                           ; переместил dx в стек
      int 21h                           ; прочитал строку
      pop bp                            ; bp = dx
      xor bx, bx                  ; bx = 0
      mov bl, ds:[bp + 1]         ; bl = фактический размер строки
      add bx, bp                  ; bx указывает на end-2, конец строки
      add bx, 2
      mov word ptr [bx + 1], 240ah ;24h в конце строки

      ;востановил регистры
      pop bx
      pop bp
      pop ax
      ret
ReadString ENDP

RestoreInterruptions PROC

      push ax
      push ds
      push dx

      cmp overwritten, 0                ; проверил на установленные прерывания
      jz already_restored
```

```
      push ds;восстановил 60h

      mov dx, KEEP_IP_60h               ; сохраняю в dx, ax ip и cs от 60h
      mov ax, KEEP_CS_60h
      mov ds, ax                        ; ds=ax
      mov al, 60h                       ; число векторов
      mov ah, 25h                       ;функция для перезаписи
      int 21h                                 ; запускаю

      pop ds

      ;восстановил 1ch
      push ds

      mov dx, KEEP_IP_9h                ; сохраняю в dx, ax ip и cs от 9h
      mov ax, KEEP_CS_9h
      mov ds, ax                        ; ds=ax
      mov al, 9h
      mov ah, 25h
      int 21h                                 ; запускаю

      pop ds

      mov overwritten, 0

already_restored:

      pop dx
      pop ds
      pop ax
      ret
RestoreInterruptions ENDP

ChangeInterruptions PROC

      push ax
      push bx
      push es

      cmp overwritten, 1            ; прерывание записано - ничего не делаю
      je already_changed            ; перехожу в already_changed


      mov cx, 0eh
      mov dx,     0ffffh
      mov ah,     86h
      int 15h

      ; copy original 9h
      mov ah, 35h                   ; 35h -  функция для получения CS:IP вектора
      mov al, 9h                    ; 9h - номер вектроа
      int 21h                             ; bx = ip 1ch, es = cs 1ch
      mov KEEP_IP_9h, bx            ;сохранил исходный адрес ip для 9h
      mov KEEP_CS_9h, es            ; сохранил исходный адрес cs для 9h

      ; change 9h to IRS_9h
      push ds

      mov dx, offset IRS_9h   ; Загружаю в dx адрес IRS_9h
      mov ax, seg IRS_9h            ; Загружаю в ax сегмент IRS_9h
```

```
      mov ds, ax                    ;  Устанавливаю ds равным ax (сегменту
IRS_9h)
      mov al, 9h                    ; 9h это номер вектора
      mov ah, 25h                   ; 25h - это функция для перезаписи вектора
      int 21h                             ; Перезаписываю

      pop ds

      ; копирую оригинальный 60h
      mov ah, 35h                   ; 35h - это функция для получения CS:IP
вектора
      mov al, 60h                   ; 60h - это номер вектора
      int 21h                             ; bx = ip, es = cs
      mov KEEP_IP_60h, bx           ; Запоминаю исходный ip 60h
      mov KEEP_CS_60h, es           ; Запоминаю исходный cs 60h

      ; меняю 60h на IRS_60h
      push ds

      mov dx, offset IRS_60h  ; загружаю в dx адрес IRS_60h
      mov ax, seg IRS_60h           ; загружаю в ax сегмент IRS_60h
      mov ds, ax                    ; Устанавливаю ds равным ax (сегменту
IRS_60h)
      mov al, 60h                   ; 60h - это номер вектора
      mov ah, 25h                   ; 25h - это функция для перезаписи вектора
      int 21h                             ; Перезаписываю

      pop ds

      ; переписаны прерывания
      mov overwritten, 1

already_changed:

      ;восстанавливаю регистры
      pop es
      pop bx
      pop ax
      ret
ChangeInterruptions ENDP

IRS_60h PROC
      ; сохраняю регистры
      push ax
      push si
      push di
      push dx

      cmp called, 0
      je end_of_irs

      ; 9h вызвано - выполняется действие
      mov ax, DATA
      mov dx, ax
      mov es, ax

      lea si, original
      lea di, processed

      cld ; move forward

      iterate:
            lodsb
```

```
            cmp al, '$'
            je end_of_str
            cmp al, '0'
            jl next
            cmp al, '9'
            jg next

            stosb
            next:
                  jmp iterate

end_of_str:
      xor ax, ax
      mov al, header[1]
      mov si, ax

      iterate_backwards:
            dec si
            cmp si, -1
            je success
            mov al, original[si]
            cmp al, '0'
            jl ok
            cmp al, '9'
            jg ok
            jmp next_back
            ok:
            stosb
            next_back:
                  jmp iterate_backwards

success:
      mov al, '$'
      stosb
      mov dx, offset processed
      call Print

end_of_irs:
      mov al, 20h
      out 20h, al

      ;восстанавливаю регистры
      pop dx
      pop di
      pop si
      pop ax
      iret                                    iret восстановит ip, cs и флаги
IRS_60h ENDP

IRS_9h PROC
      mov called, 1
      jmp dword ptr [KEEP_IP_9h]          ; вызываю оригинальный 9h
IRS_9h ENDP

Main PROC FAR
      ;инициализирую
      push ds
      sub ax, ax ; ax = 0
      push ax ; сохраняю смещение 0
      mov ax, DATA ; ax = данные
      mov ds, ax ; инициализирую ds
      mov es, ax ; инициализирую es
```

```
      ; Вывод приветственной строки
      mov dx, offset greeting ;сохраняю смещение
      call Print                    ; вызываю процедуру Print

      ;считывание пользовательской строки
      mov dx, offset header
      call ReadString               ;сохранил смещение, вызвал процедуру

      ;печать оригинальной строки
      mov dx, offset original
      call Print

      ;изменения 9h и 60h
      call ChangeInterruptions

      ;задержка 5 секунд
      mov cx, 4Dh
      mov dx,     0ffffh
      mov ah,     86h
      int 15h

      cmp overwritten, 0                              ; проверка, все еще ли
перезаписано 60h
      jz done                                                 ; не можем вызывать
60h по умолчанию

      ; Прерывание 60h может быть вызвано только если прерывания не были
восстановлены

      int 60h                                                 ; вызов
пользовательского прерывания

      done:

      ; Восстановление прерываний в любом случае
      call RestoreInterruptions                ; восстановление прерываний
      ret
Main ENDP
CODE ENDS
     END Main
```