```
IDEAL
RADIX   16
P286
MODEL   LARGE

include 'tiny-os.inc'

STACK   STACK_SIZE

DATASEG
DSEG_BEG = THIS WORD
        real_ss dw      ?
        real_sp dw      ?
        real_es dw      ?

GDT_BEG = $
LABEL   gdtr            WORD
gdt_0   desc_struc <0,0,0,0,0>
gdt_gdt         desc_struc <GDT_SIZE-1,,,DATA_ACC,0>
gdt_idt         desc_struc <IDT_SIZE-1,,,IDT_ACC,0>
gdt_ds  desc_struc <DSEG_SIZE-1,,,DATA_ACC,0>
gdt_cs  desc_struc <CSEG_SIZE-1,,,CODE_ACC,0>
gdt_ss  desc_struc <STACK_SIZE-1,,,DATA_ACC,0>
gdt_bio         desc_struc <B_DATA_SIZE-1,B_DATA_ADDR,0,DATA_ACC,0>
gdt_crt         desc_struc <CRT_SIZE-1,CRT_LOW,CRT_SEG,DATA_ACC,0>
gdt_mda         desc_struc <MONO_SIZE-1,MONO_LOW,CRT_SEG,DATA_ACC,0>
GDT_SIZE = ($ - GDT_BEG)

; Область памяти для загрузки регистра IDTR
idtr    idtr_struc      <IDT_SIZE,,,0>

; Таблица дескрипторов прерываний
IDT_BEG = $
; ---------------------- Вентили исключений --------------------
idt     idt_struc <OFFSET exc_00,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_01,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_02,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_03,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_04,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_05,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_06,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_07,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_08,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_09,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_0A,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_0B,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_0C,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_0D,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_0E,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_0F,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_10,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_11,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_12,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_13,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_14,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_15,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_16,CS_DESCR,0,TRAP_ACC,0>
```

```
        idt_struc <OFFSET exc_17,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_18,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_19,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_1A,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_1B,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_1C,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_1D,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_1E,CS_DESCR,0,TRAP_ACC,0>
        idt_struc <OFFSET exc_1F,CS_DESCR,0,TRAP_ACC,0>
; --------------- Вентили аппаратных прерываний ---------------
; int 20h-IRQ0
        idt_struc <OFFSET Timer_int,CS_DESCR,0,INT_ACC,0>
; int 21h-IRQ1
        idt_struc <OFFSET Keyb_int,CS_DESCR,0,INT_ACC,0>
; int 22h, 23h, 24h, 25h, 26h, 27h-IRQ2-IRQ7
        idt_struc 6 dup (<OFFSET dummy_iret0,CS_DESCR,0,INT_ACC,0>)
; int 28h, 29h, 2ah, 2bh, 2ch, 2dh, 2eh, 2fh-IRQ8-IRQ15
        idt_struc 8 dup (<OFFSET dummy_iret1,CS_DESCR,0,INT_ACC,0>)
; -------------------- Вентиль прерывания --------------------
; int 30h
idt_struc       <OFFSET Int_30h_Entry,CS_DESCR,0,INT_ACC,0>
IDT_SIZE        = ($ - IDT_BEG)

CODESEG
PROC    start
        mov     ax,DGROUP
        mov     ds,ax
        call    set_crt_base
        mov     bh, 77h
        call    clrscr
; Устанавливаем защищённый режим
        call    set_pmode
        call    write_hello_msg
; Размаскируем прерывания от таймера и клавиатуры
        in      al,INT_MASK_PORT
        and     al,0fch
        out     INT_MASK_PORT,al
; Ожидаем нажатия на клавишу <ESC>
charin:
        int     30h     ; ожидаем нажатия на клавишу
                        ; AX - скан-код клавиши,
                        ; BX - состояние переключающих клавиш
        cmp     al, 1   ; если <ESC> - выход из цикла
        jz      continue

        push    bx              ; выводим скан-код на экран
        mov     bx, 0301h       ; координаты вывода
        call    Print_Word
        pop     bx

        mov     ax, bx          ; выводим состояние
        push    bx              ; переключающих клавиш
        mov     bx, 0306h
        call    Print_Word
        pop     bx

        jmp     charin
```

```
; Следующий байт находится в сегменте кода. Он используется для
; демонстрации возникновения исключения при попытке записи в сегмент
; кода.
wrong1  db ?

continue:
; После нажатия на клавишу <ESC> выходим в это место программы.
; Следующие несколько строк демонстрируют команды, которые вызывают
; исключение.
; Попытка записи за конец сегмента данных. Метка wrong находится в
; самом конце программы.
        mov     [wrong], al
; Попытка записи в сегмент кода.
;       mov     [wrong1], al
; Попытка извлечения из пустого стека.
;       pop     ax
; Загрузка в сегментный регистр неправильного селектора.
;       mov     ax, 1280h
;       mov     ds, ax
; Прямой вызов исключения при помощи команды прерывания.
;       int     1


        call    set_rmode       ; установка реального режима

        mov     bh, 07h         ; стираем экран и
        call    clrscr          ; выходим в DOS
        mov     ah,4c
        int     21h

ENDP    start

MACRO setgdtentry
        mov     [(desc_struc bx).base_l],ax
        mov     [(desc_struc bx).base_h],dl
ENDM

; -----------------------------------------
; Установка защищённого режима
; -----------------------------------------
PROC    set_pmode       NEAR
        mov     ax,DGROUP
        mov     dl,ah
        shr     dl,4
        shl     ax,4
        mov     si,ax
        mov     di,dx
        add     ax,OFFSET gdtr
        adc     dl,0
        mov     bx,OFFSET gdt_gdt
        setgdtentry

; Заполняем дескриптор в GDT, указывающий на дескрипторную таблицу
; прерываний
        mov     ax,si   ; загружаем 24-битовый адрес сегмента
        mov     dx,di   ; данных
```

```
        add     ax,OFFSET idt   ; адрес дескриптора для IDT
        adc     dl,0
        mov     bx,OFFSET gdt_idt
        setgdtentry

; Заполняем структуру для загрузки регистра IDTR
        mov     bx,OFFSET idtr
        mov     [(idtr_struc bx).idt_l],ax
        mov     [(idtr_struc bx).idt_h],dl

        mov     bx,OFFSET gdt_ds
        mov     ax,si
        mov     dx,di
        setgdtentry

        mov     bx,OFFSET gdt_cs
        mov     ax,cs
        mov     dl,ah
        shr     dl,4
        shl     ax,4
        setgdtentry

        mov     bx,OFFSET gdt_ss
        mov     ax,ss
        mov     dl,ah
        shr     dl,4
        shl     ax,4
        setgdtentry

; готовим возврат в реальный режим
        push    ds
        mov     ax,40
        mov     ds,ax
        mov     [WORD 67],OFFSET shutdown_return
        mov     [WORD 69],cs
        pop     ds

        cli
        mov     al,8f
        out     CMOS_PORT,al
        jmp     del1
del1:
        mov     al,5
        out     CMOS_PORT+1,al

        mov     ax,[rl_crt]     ; сегмент видеопамяти
        mov     es,ax

        call    enable_a20      ; открываем линию A20

        mov     [real_ss],ss    ; сохраняем сегментные
        mov     [real_es],es    ; регистры

; -------- Перепрограммируем контроллер прерываний --------
; Устанавливаем для IRQ0-IRQ7 номера прерываний 20h-27h
        mov     dx,MASTER8259A
        mov     ah,20h
```

```
        call    set_int_ctrlr

; Устанавливаем для IRQ8-IRQ15 номера прерываний 28h-2Fh
        mov     dx,SLAVE8259A
        mov     ah,28h
        call    set_int_ctrlr

; Загружаем регистры IDTR и GDTR
        lidt    [FWORD idtr]
        lgdt    [QWORD gdt_gdt]

; Переключаемся в защищённый режим
        mov     ax,VIRTUAL_MODE
        lmsw    ax

;       jmp     far flush
        db      0ea
        dw      OFFSET flush
        dw      CS_DESCR
LABEL   flush   FAR

; Загружаем селекторы в сегментные регистры
        mov     ax,SS_DESCR
        mov     ss,ax
        mov     ax,DS_DESCR
        mov     ds,ax

        sti
        ret
ENDP    set_pmode

; --------------------------------
; Возврат в реальный режим
; --------------------------------
DATASEG
; Пустой дескриптор для выполнения возврата процессора в реальный
; режим через перевод его в состояние отключения.
null_idt idt_struc <>

CODESEG
PROC    set_rmode       NEAR

        mov     [real_sp],sp
; Переводим процессор в состояние отключения, это эквивалентно
; аппаратному сбросу, но выполняется быстрее. Сначала мы загружаем
; IDTR нулями, затем выдаём команду прерывания.
        lidt    [FWORD null_idt]
        int     3

; Это старый способ сброса процессора через контроллер клавиатуры.
;       mov     al,SHUT_DOWN
;       out     STATUS_PORT,al

rwait:
        hlt
        jmp     rwait
```

```
LABEL   shutdown_return FAR

        in      al,INT_MASK_PORT
        and     al,0
        out     INT_MASK_PORT,al

        mov     ax,DGROUP
        mov     ds,ax
        assume  ds:DGROUP

        cli
        mov     ss,[real_ss]
        mov     sp,[real_sp]
        mov     ax,000dh
        out     CMOS_PORT,al
        sti
        mov     es,[real_es]
        call    disable_a20
        ret
ENDP    set_rmode

; ------------------------------------------------
; Обработка исключений
; ------------------------------------------------
; Обработчики исключений. Записываем в AX номер исключения и
; передаём управление процедуре shutdown

LABEL   exc_00  WORD
        mov     ax,0
        jmp     shutdown
LABEL   exc_01  WORD
        mov     ax,1
        jmp     shutdown
LABEL   exc_02  WORD
        mov     ax,2
        jmp     shutdown
LABEL   exc_03  WORD
        mov     ax,3
        jmp     shutdown
LABEL   exc_04  WORD
        mov     ax,4
        jmp     shutdown
LABEL   exc_05  WORD
        mov     ax,5
        jmp     shutdown
LABEL   exc_06  WORD
        mov     ax,6
        jmp     shutdown
LABEL   exc_07  WORD
        mov     ax,7
        jmp     shutdown
LABEL   exc_08  WORD
        mov     ax,8
        jmp     shutdown
LABEL   exc_09  WORD
        mov     ax,9
        jmp     shutdown
```

```
LABEL   exc_0A  WORD
        mov     ax,0ah
        jmp     shutdown
LABEL   exc_0B  WORD
        mov     ax,0bh
        jmp     shutdown
LABEL   exc_0C  WORD
        mov     ax,0ch
        jmp     shutdown
LABEL   exc_0D  WORD
        mov     ax,0dh
        jmp     shutdown
LABEL   exc_0E  WORD
        mov     ax,0eh
        jmp     shutdown
LABEL   exc_0F  WORD
        mov     ax,0fh
        jmp     shutdown
LABEL   exc_10  WORD
        mov     ax,10h
        jmp     shutdown
LABEL   exc_11  WORD
        mov     ax,11h
        jmp     shutdown
LABEL   exc_12  WORD
        mov     ax,12h
        jmp     shutdown
LABEL   exc_13  WORD
        mov     ax,13h
        jmp     shutdown
LABEL   exc_14  WORD
        mov     ax,14h
        jmp     shutdown
LABEL   exc_15  WORD
        mov     ax,15h
        jmp     shutdown
LABEL   exc_16  WORD
        mov     ax,16h
        jmp     shutdown
LABEL   exc_17  WORD
        mov     ax,17h
        jmp     shutdown
LABEL   exc_18  WORD
        mov     ax,18h
        jmp     shutdown
LABEL   exc_19  WORD
        mov     ax,19h
        jmp     shutdown
LABEL   exc_1A  WORD
        mov     ax,1ah
        jmp     shutdown
LABEL   exc_1B  WORD
        mov     ax,1bh
        jmp     shutdown
LABEL   exc_1C  WORD
        mov     ax,1ch
        jmp     shutdown
```

```
LABEL   exc_1D  WORD
        mov     ax,1dh
        jmp     shutdown
LABEL   exc_1E  WORD
        mov     ax,1eh
        jmp     shutdown
LABEL   exc_1F  WORD
        mov     ax,1fh
        jmp     shutdown

DATASEG
exc_msg db "Exception ...., ....:.... code ..... Press any key... "

CODESEG

; ------------------------------------------------
; Вывод на экран номера исключения, кода ошибки, дампа регистров и
; возврат в реальный режим.
; ------------------------------------------------

PROC    shutdown        NEAR
        call    rdump   ; дамп регистров процессора
        push    ax

; Выводим сообщение об исключении
        mov     ax,[vir_crt]
        mov     es,ax
        mov     bx,1d
        mov     ax,4
        mov     si,OFFSET exc_msg
        mov     dh,74h
        mov     cx, SIZE exc_msg
        call    writexy
        pop     ax

        mov     bx, 040bh       ; номер исключения
        call    Print_Word

        pop     ax
        mov     bx, 0420h       ; код ошибки
        call    Print_Word

        pop     ax
        mov     bx, 0416h       ; смещение
        call    Print_Word

        pop     ax
        mov     bx, 0411h       ; селектор
        call    Print_Word

        call    set_rmode       ; возвращаемся в реальный режим

        mov     ax, 0   ; ожидаем нажатия на клавишу
        int     16h

        mov     bh, 07h
        call    clrscr
```

```
        mov     ah,4Ch
        int     21h

ENDP    shutdown

; ------------------------------------------------
; Перепрограммирование контроллера прерываний
;       На входе: DX - порт контроллера прерывания
;                 AH - начальный номер прерывания
; ------------------------------------------------

PROC    set_int_ctrlr   NEAR
        mov     al,11
        out     dx,al
        jmp     SHORT $+2
        mov     al,ah
        inc     dx
        out     dx,al
        jmp     SHORT $+2
        mov     al,4
        out     dx,al
        jmp     SHORT $+2
        mov     al,1
        out     dx,al
        jmp     SHORT $+2
        mov     al,0ff
        out     dx,al
        dec     dx
        ret
ENDP    set_int_ctrlr

; -------------------------------
; Разрешение линии A20
; -------------------------------
PROC    enable_a20      NEAR
        mov     al,A20_PORT
        out     STATUS_PORT,al
        mov     al,A20_ON
        out     KBD_PORT_A,al
        ret
ENDP    enable_a20

; -------------------------------
; Запрещение линии A20
; -------------------------------
PROC    disable_a20     NEAR
        mov     al,A20_PORT
        out     STATUS_PORT,al
        mov     al,A20_OFF
        out     KBD_PORT_A,al
        ret
ENDP    disable_a20

; ---------- Обработчик аппаратных прерываний IRQ2-IRQ7
PROC    dummy_iret0      NEAR
        push    ax
```

```
; Посылаем сигнал конца прерывания в первый контроллер 8259A
        mov     al,EOI
        out     MASTER8259A,al
        pop     ax
        iret
ENDP    dummy_iret0

; ---------- Обработчик аппаратных прерываний IRQ8-IRQ15
PROC    dummy_iret1      NEAR
        push    ax

; Посылаем сигнал конца прерывания в первый и второй контроллеры
; 8259A
        mov     al,EOI
        out     MASTER8259A,al
        out     SLAVE8259A,al
        pop     ax
        iret
ENDP    dummy_iret1

; -------------------------------------------------
; Процедура задерживает выполнение программы на некоторое время,
; зависящее от быстродействия процессора.
; -------------------------------------------------
PROC    pause           NEAR
        push    cx
        mov     cx,10
ploop0:
        push    cx
        xor     cx,cx
ploop1:
        loop    ploop1
        pop     cx
        loop    ploop0
        pop     cx
        ret
ENDP    pause

; ----------------------------------------
;       Процедуры для работы с клавиатурой
; ----------------------------------------
DATASEG
        key_flag        db      0
        key_code        dw      0
        ext_scan        db      0
        keyb_status     dw      0

CODESEG
; ----------------------------------------------
; Обработчик аппаратного прерывания клавиатуры
; ----------------------------------------------
PROC    Keyb_int         NEAR
        push    ax
        mov     al, [ext_scan]  ; расширенный скан-код
        cmp     al, 0           ; или обычный ?
        jz      normal_scan1
```

```
; --------- обработка расширенного скан-кода -------------
        cmp     al, 0e1h                ; это клавиша <Pause>?
        jz      pause_key

        in      al, 60h         ; вводим скан-код

        cmp     al, 2ah         ; игнорируем префикс 2Ah
        jz      intkeyb_exit_1

        cmp     al, 0aah                ; игнорируем отпускание
        jz      intkeyb_exit_1  ; клавиш

        mov     ah, [ext_scan]  ; записываем скан-код и
        call    Keyb_PutQ               ; расширенный скан-код
                                        ; в "очередь", состоящую
                                        ; из одного слова

        mov     al, 0           ; сбрасываем признак
        mov     [ext_scan], al  ; получения расширенного
        jmp     intkeyb_exit    ; скан-кода

pause_key:                      ; обработка клавиши <Pause>

        in      al, 60h         ; вводим скан-код
        cmp     al, 0c5h                ; если это код <Pause>,
        jz      pause_key1      ; записываем его в очередь,
        cmp     al, 45h         ; иначе игнорируем
        jz      pause_key1
        jmp     intkeyb_exit

pause_key1:
        mov     ah, [ext_scan]  ; запись в очередь
        call    Keyb_PutQ               ; кода клавиши <Pause>

        mov     al, 0           ; сбрасываем признак
        mov     [ext_scan], al  ; получения расширенного
        jmp     intkeyb_exit    ; скан-кода

; --------- обработка обычного скан-кода -------------
normal_scan1:

        in      al, 60h         ; вводим скан-код

        cmp     al, 0feh                ; игнорируем FEh
        jz      intkeyb_exit

        cmp     al, 0e1h                ; расширенный скан-код?
        jz      ext_key         ; если да, то на обработку
                                        ; расширенного скан-кода
        cmp     al, 0e0h
        jnz     normal_scan

ext_key:
        mov     [ext_scan], al  ; устанавливаем признак
        jmp     intkeyb_exit    ; расширенного скан-кода

; Сброс признака расширенного скан-кода и выход
```

```
intkeyb_exit_1:
        mov     al, 0
        mov     [ext_scan], al
        jmp     intkeyb_exit

; Запись нормального скан-кода в очередь и выход
normal_scan:
        mov     ah, 0
        call    Keyb_PutQ

intkeyb_exit:
        in      al, 61h         ; разблокируем клавиатуру
        mov     ah, al
        or      al, 80h
        out     61h, al
        xchg    ah, al
        out     61h, al

        mov     al,EOI          ; посылаем сигнал конца
        out     MASTER8259A,al  ; прерывания

        pop     ax
        sti
        iret
ENDP    Keyb_int

; ------------------------------------------------
; Запись скан-кода и расширенного скан-кода в "буфер", состоящий из
; одного слова.
; ------------------------------------------------
PROC    Keyb_PutQ       NEAR
        push    ax
        mov     [key_code], ax  ; записываемый код

; ------- Обрабатываем переключающие клавиши ---------
        cmp     ax, 002ah               ; L_SHIFT down
        jnz     @@kb1
        mov     ax, [keyb_status]
        or      ax, L_SHIFT
        mov     [keyb_status], ax
        jmp     keyb_putq_exit
@@kb1:
        cmp     ax, 00aah               ; L_SHIFT up
        jnz     @@kb2
        mov     ax, [keyb_status]
        and     ax, NL_SHIFT
        mov     [keyb_status], ax
        jmp     keyb_putq_exit
@@kb2:
        cmp     ax, 0036h               ; R_SHIFT down
        jnz     @@kb3
        mov     ax, [keyb_status]
        or      ax, R_SHIFT
        mov     [keyb_status], ax
        jmp     keyb_putq_exit
@@kb3:
        cmp     ax, 00b6h               ; R_SHIFT up
```

```
        jnz     @@kb4
        mov     ax, [keyb_status]
        and     ax, NR_SHIFT
        mov     [keyb_status], ax
        jmp     keyb_putq_exit
@@kb4:
        cmp     ax, 001dh               ; L_CTRL down
        jnz     @@kb5
        mov     ax, [keyb_status]
        or      ax, L_CTRL
        mov     [keyb_status], ax
        jmp     keyb_putq_exit
@@kb5:
        cmp     ax, 009dh               ; L_CTRL up
        jnz     @@kb6
        mov     ax, [keyb_status]
        and     ax, NL_CTRL
        mov     [keyb_status], ax
        jmp     keyb_putq_exit
@@kb6:
        cmp     ax, 0e01dh      ; R_CTRL down
        jnz     @@kb7
        mov     ax, [keyb_status]
        or      ax, R_CTRL
        mov     [keyb_status], ax
        jmp     keyb_putq_exit
@@kb7:
        cmp     ax, 0e09dh      ; R_CTRL up
        jnz     @@kb8
        mov     ax, [keyb_status]
        and     ax, NR_CTRL
        mov     [keyb_status], ax
        jmp     keyb_putq_exit
@@kb8:
        cmp     ax, 0038h               ; L_ALT down
        jnz     @@kb9
        mov     ax, [keyb_status]
        or      ax, L_ALT
        mov     [keyb_status], ax
        jmp     keyb_putq_exit
@@kb9:
        cmp     ax, 00b8h               ; L_ALT up
        jnz     @@kb10
        mov     ax, [keyb_status]
        and     ax, NL_ALT
        mov     [keyb_status], ax
        jmp     keyb_putq_exit
@@kb10:
        cmp     ax, 0e038h      ; R_ALT down
        jnz     @@kb11
        mov     ax, [keyb_status]
        or      ax, R_ALT
        mov     [keyb_status], ax
        jmp     keyb_putq_exit
@@kb11:
        cmp     ax, 0e0b8h      ; R_ALT up
        jnz     @@kb12
```

```
        mov     ax, [keyb_status]
        and     ax, NR_ALT
        mov     [keyb_status], ax
        jmp     keyb_putq_exit
@@kb12:
        cmp     ax, 003ah               ; CAPS_LOCK up
        jnz     @@kb13
        mov     ax, [keyb_status]
        xor     ax, CAPS_LOCK
        mov     [keyb_status], ax
        jmp     keyb_putq_exit
@@kb13:
        cmp     ax, 00bah               ; CAPS_LOCK down
        jnz     @@kb14
        jmp     keyb_putq_exit
@@kb14:
        cmp     ax, 0046h               ; SCR_LOCK up
        jnz     @@kb15
        mov     ax, [keyb_status]
        xor     ax, SCR_LOCK
        mov     [keyb_status], ax
        jmp     keyb_putq_exit
@@kb15:
        cmp     ax, 00c6h               ; SCR_LOCK down
        jnz     @@kb16
        jmp     keyb_putq_exit
@@kb16:
        cmp     ax, 0045h               ; NUM_LOCK up
        jnz     @@kb17
        mov     ax, [keyb_status]
        xor     ax, NUM_LOCK
        mov     [keyb_status], ax
        jmp     keyb_putq_exit
@@kb17:
        cmp     ax, 00c5h               ; NUM_LOCK down
        jnz     @@kb18
        jmp     keyb_putq_exit
@@kb18:
        cmp     ax, 0e052h      ; INSERT up
        jnz     @@kb19
        mov     ax, [keyb_status]
        xor     ax, INSERT
        mov     [keyb_status], ax
        jmp     keyb_putq_exit
@@kb19:
        cmp     ax, 0e0d2h      ; INSERT down
        jnz     @@kb20
        jmp     keyb_putq_exit
@@kb20:
        test    ax, 0080h               ; фильтруем отжатия клавиш
        jnz     keyb_putq_exit
        mov     al, 0ffh                ; устанавиваем признак
        mov     [key_flag], al  ; готовности для чтения
                                        ; символа из "буфера"
keyb_putq_exit:
        pop     ax
        ret
```

```
ENDP    Keyb_PutQ

; ------------------------------------------------
; Программное прерывание, предназначенное для чтения символа из
; буфера клавиатуры. По своим функциям напоминает прерывание INT 16h
; реального режима. В AX возвращается скан-код нажатой клавиши,
; в BX - состояние переключающих клавиш.
; ------------------------------------------------
PROC    Int_30h_Entry    NEAR
        push    dx                      ; запрещаем прерывания и
        cli                             ; сбрасываем признак
        mov     al, 0           ; готовности скан-кода
        mov     [key_flag], al  ; в буфере клавиатуры

; Ожидаем прихода прерывания от клавиатуры. Процедура клавиатурного
; прерывания установит признак в переменной key_flag.
keyb_int_wait:
        sti                             ; разрешаем прерывания
        nop                             ; ждём прерывание
        nop
        cli                             ; запрещаем прерывания
        mov     al, [key_flag]  ; и опрашиваем флаг
        cmp     al, 0           ; готовности скан-кода
        jz      keyb_int_wait

        mov     al, 0           ; сбрасываем флаг
        mov     [key_flag], al  ; готовности

        mov     ax, [key_code]           ; записываем скан-код
        mov     bx, [keyb_status]       ; и состояние переключающих
                                                ; клавиш

        sti                             ; разрешаем прерывания
        pop     dx
        iret
ENDP    Int_30h_Entry

; ----------------------------------------
;       TIMER section
; ----------------------------------------
DATASEG
        timer_cnt dw    0

CODESEG
PROC    Timer_int        NEAR
        cli
        push    ax

; Увеличиваем содержимое счётчика времени
        mov     ax, [timer_cnt]
        inc     ax
        mov     [timer_cnt], ax

timer_exit:
; Посылаем команду конца прерывания
        mov     al,EOI
        out     MASTER8259A,al
```

```
        pop     ax
        sti
        iret
ENDP    Timer_int

; -------------------------------------------------
; Процедуры обслуживания видеоконтроллера
; -------------------------------------------------
DATASEG
        columns db      80d
        rows    db      25d
        rl_crt  dw      COLOR_SEG
        vir_crt dw      CRT_DESCR
        curr_line       dw      0d
        text_buf        db      "           "

CODESEG
; ----------------------------------------
; Определение адреса видеопамяти
; ----------------------------------------
PROC    set_crt_base    NEAR
        mov     ax,40
        mov     es,ax
        mov     bx,[WORD es:4a]
        mov     [columns],bl
        mov     bl,[BYTE es:84]
        inc     bl
        mov     [rows],bl
        mov     bx,[WORD es:PORT_6845]
        cmp     bx,COLOR_PORT
        je      color_crt
        mov     [rl_crt],MONO_SEG
        mov     [vir_crt],MDA_DESCR
color_crt:
        ret
ENDP    set_crt_base

; -----------------------------------
; Запись строки в видеопамять
; -----------------------------------
PROC    writexy         NEAR
        push    si
        push    di
        mov     dl,[columns]
        mul     dl
        add     ax,bx
        shl     ax,1
        mov     di,ax
        mov     ah,dh
write_loop:
        lodsb
        stosw
        loop    write_loop
        pop     di
        pop     si
        ret
```

```
ENDP    writexy

; --------------------------------------
; Стирание экрана (в реальном режиме)
; --------------------------------------
PROC    clrscr          NEAR
        xor     cx,cx
        mov     dl,[columns]
        mov     dh,[rows]
        mov     ax,0600
        int     10
        ret
ENDP    clrscr

DATASEG
hello_msg db " Protected mode monitor *TINY/OS*, for CPU 80286"

CODESEG
; ------------------------------------
; Вывод начального сообщения в защищённом режиме
; ------------------------------------
PROC    write_hello_msg NEAR
        mov     ax,[vir_crt]
        mov     es,ax
        mov     si,OFFSET hello_msg
        mov     bx,0
        mov     ax,[curr_line]
        inc     [curr_line]
        mov     cx,SIZE hello_msg
        mov     dh,30h
        call    writexy
        ret
ENDP    write_hello_msg

; ------------------------------------------------
; Процедура выводит на экран содержимое AX
;       (x,y) = (bh, bl)
; ------------------------------------------------
PROC Print_Word near
        push ax
        push bx
        push dx

        push ax
        mov cl,8
        rol ax,cl
        call Byte_to_hex
        mov     [text_buf], dh
        mov     [text_buf+1], dl

        pop ax
        call Byte_to_hex
        mov     [text_buf+2], dh
        mov     [text_buf+3], dl

        mov     si, OFFSET text_buf
        mov     dh, 70h
```

```
        mov     cx, 4
        mov     al, bh
        mov     ah, 0

        mov     bh, 0
        call    writexy

        pop dx
        pop bx
        pop ax
        ret
ENDP Print_Word

DATASEG
tabl db '0123456789ABCDEF'

CODESEG
; ----------------------------------------
; Преобразование байта в шестнадцатеричный символьный формат
; al - входной байт
; dx - выходное слово
; ----------------------------------------
PROC Byte_to_hex near
        push    cx
        push    bx

        mov     bx, OFFSET tabl

        push    ax
        and     al,0fh
        xlat
        mov     dl,al

        pop     ax
        mov     cl,4
        shr     al,cl
        xlat
        mov     dh,al

        pop bx
        pop cx
        ret

ENDP Byte_to_hex

DATASEG
reg_title  db " CS   IP   AX   BX   CX   DX   SP   BP   SI   DI"
;                  ....  ....  ....  ....  ....  ....  ....  ....
sreg_title db " DS   ES   SS   FLAGS                          "
;                  ....  ....  ....  ....  ....  ....  ....  ....

CODESEG
; ----------------------------------------------
; Вывод на экран содержимого регистров процессора
; ----------------------------------------------
PROC    rdump NEAR
        pushf
```

```
        pusha

        mov     di, es

        mov     ax,[vir_crt]
        mov     es,ax
        mov     si,OFFSET reg_title
        mov     bx,1                    ; (X,Y) = (AX,BX)
        mov     ax,6
        mov     cx,SIZE reg_title
        mov     dh,1fh                  ; чёрный на голубом фоне
        call    writexy

; Выводим содержимое всех регистров
        mov     ax,cs           ; cs
        mov     bx, 0702h
        call    Print_Word

        mov     bp, sp

        mov     ax, [bp+18d]     ; ip
        mov     bx, 0708h
        call Print_Word

        mov     bx, 070eh
        mov     ax,[bp+14d]     ; ax
        call    Print_Word

        mov     bx, 0714h
        mov     ax,[bp+8d]      ; bx
        call Print_Word

        mov     bx, 071ah
        mov     ax,[bp+12d]     ; cx
        call Print_Word

        mov     bx, 0720h
        mov     ax,[bp+10d]     ; dx
        call Print_Word

        mov     ax,bp
        add     ax,20d          ; sp
        mov     bx, 0726h
        call Print_Word

        mov     ax,[bp+4d]      ; bp
        mov     bx, 072ch
        call Print_Word

        mov     bx, 0732h
        mov     ax,[bp+2]       ; si
        call Print_Word

        mov     bx, 0738h
        mov     ax, [bp]        ; di
        call Print_Word
```

```
        mov     si,OFFSET sreg_title
        mov     bx,1
        mov     ax,8
        mov     cx,SIZE sreg_title
        mov     dh,1fh
        call    writexy

        mov     bx, 0902h
        mov     ax, ds          ; ds
        call Print_Word

        mov     bx, 0908h
        mov     ax, di          ; es
        call Print_Word

        mov     bx, 090eh
        mov     ax,ss           ; ss
        call Print_Word

        mov     bx, 0914h
        mov     ax, [bp+16d]    ; flags
        call Print_Word

; Восстанавливаем содержимое регистров
        popa
        popf
        ret
ENDP    rdump

CSEG_SIZE       = ($ - start)

DATASEG
DSEG_SIZE       = ($ - DSEG_BEG)

wrong   db      ?
        END     start
```