Пример №3 – «Ядро крошечной операционной системы – Kernel».

В этом примере «параллельно» выполняются две задачи, их выполнение можно наблюдать на экране. Так же обрабатывается прерывание от клавиатуры: скан-код нажатой клавиши выводится на экране.

Ниже приведен исходный текст программы (сначала главный модуль, а за ним «включаемые» *.inc - модули).

kernel.asm

```
        .386P
        .MODEL   LARGE

        INCLUDE MACROSES.INC                ; MAIN DEFINITIONS

SIZE_STACK      EQU     1000H

;---------------------------------------;
; 16-BIT REAL/PROTECTED NODE CODE SEGMENT ;
;---------------------------------------;

CODE16  SEGMENT PARA USE16
CODE16_BEGIN    =       $
        ASSUME          CS:CODE16,DS:DATA,ES:DATA
```

```
START:
        MOV     AX, DATA
        MOV     DS, AX
        MOV     ES, AX                  ; LOAD DATA SEGMENTS

        LEA     DX, MSG_STARTUP
        MOV     AH, 9
        INT     21H

;       MOV     EAX, CR0                ; DOESN'T WORK UNDER WIN :(
        SMSW    AX                      ; BUT THIS WORKS PERFECTLY :)
        TEST    AL, 1                   ; IS PROTECTED MODE ACTIVE?
        JZ      START_NO_PM             ; NO, IT ISN'T
        LEA     DX, MSG_VM
        MOV     AH, 9
        INT     21H
        JMP     START_EXIT              ; YES, IT IS. EXITING...

START_NO_PM:
        MOV     EAX, 3
        CALL    delayRTC

        @ENABLE_A20                     ; ENABLE A20 LINE
        IN      AL, PORT_INT_MASK_M
        MOV     INT_MASK_M, AL          ; SAVE MASK OF HARDWARE
INTS AT MASTER CNTRLR
        IN      AL, PORT_INT_MASK_S
        MOV     INT_MASK_S, AL          ; SAVE MASK OF HARDWARE
INTS AT SLAVE CNTRLR
        CLI                     ; DISABLE MASKABLE INTS
        @DISABLE_NMI                    ; DISABLE NON-MASKABLE INTS

        INCLUDE SETUPGDT.INC            ; FILL GDT
        INCLUDE SETUPIDT.INC            ; FILL IDT
        LGDT    FWORD PTR GDT_GDT       ; LOAD GDTR
        LIDT    FWORD PTR IDTR          ; LOAD IDTR
```

```
    MOV     EAX, CR0
    OR      AL, 1
    MOV     CR0, EAX                ; SET UP PROTECTED MODE

    @JUMP                           ; OVERLOAD CODE SELECTOR
    MOV     AX, DS_DESC
    MOV     DS, AX                  ; LOAD DS WITH DATA SELECTOR
    MOV     ES, AX                  ; LOAD ES WITH DATA SELECTOR
    MOV     AX, SS_DESC
    MOV     SS, AX                  ; LOAD SS WITH STACK SELECTOR

    XOR     AX, AX
    MOV     FS, AX                  ; LOAD FS WITH 0
    MOV     GS, AX                  ; LOAD GS WITH 0
    LLDT    AX                      ; LOAD LDTR WITH 0

    MOVZX   ESP, SP                 ; EXPAND SP TO ESP
    PUSH    CS                      ; PREPARE FOR RETURNING BACK
    PUSH    OFFSET START_BACK_TO_16     ; TO 16-BIT PM CODE
    MOV     DS:[ESP32], ESP

    LEA     EDI, ENTER_32
    MOV     EAX, CS32_DESC
    PUSH    EAX
    PUSH    EDI                     ; PUSH 32-BIT CODE ADDRESS

    INCLUDE SETUPTSS.INC            ; FILL TSSs
    MOV     AX, TSS_MAIN_DESC
    LTR     AX                      ; LOAD TR REGISTER

    @SET_INT_CTRLR 20H, 28H         ; REINITIALISE MASTER &
SLAVE INT CONTROLLERS
    MOV     AL, 0                   ; MASK = 0
    OUT     PORT_INT_MASK_M, AL     ; UNMASK INTS
    OUT     PORT_INT_MASK_S, AL     ; UNMASK INTS
    @ENABLE_NMI                     ; ENABLE NON-MASKABLE INTS
    STI                             ; ENABLE MASKABLE INTS
```

```
        @RETF                           ; "RETURN" TO 32-BIT CODE

START_BACK_TO_16:
        CLI                             ; DISABLE MASKABLE INTS
        @DISABLE_NMI                    ; DISABLE NON-MASKABLE INTS
        MOV     AX, DS_DESC
        MOV     DS, AX                  ; SAFETY IS WHAT WE NEED!!!

        INCLUDE SET_RM1.INC             ; BACK TO REAL MODE

        MOV     AL, INT_MASK_M
        OUT     PORT_INT_MASK_M, AL             ; RESTORE MASK OF
HARDWARE INTS AT MASTER CNTRLR
        MOV     AL, INT_MASK_S
        OUT     PORT_INT_MASK_S, AL             ; RESTORE MASK OF
HARDWARE INTS AT SLAVE CNTRLR
        @ENABLE_NMI                     ; ENABLE NON-MASKABLE INTS
        STI                             ; ENABLE MASKABLE INTS
        @DISABLE_A20                    ; DISABLE A20 LINE

START_EXIT:
        MOV     EAX, 3
        CALL    delayRTC

        MOV     AX, 3
        INT     10H

        LEA     DX, MSG_CONTACT
        MOV     AH, 9
        INT     21H
        MOV     AX, 4C00H
        INT     21H                     ; EXIT TO DOS

delayRTC PROC NEAR
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Input : eax = number of seconds for delay ;;
```

```
;; Output: None, registers are preserved     ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        PUSH        EAX
        PUSH        EBX
        PUSH        ECX

        MOV         ECX, EAX
        XOR         AL, AL
        OUT         70H, AL
        JMP         DRL1
DRL1:
        IN          AL, 71H
        MOV         BL, AL
DRLL:
        XOR         AL, AL
        OUT         70H, AL
        JMP         DRL2
DRL2:
        IN          AL, 71H
        CMP         AL, BL
        JE          DRLL
        MOV         BL, AL
        LOOP        DRLL

        POP         ECX
        POP         EBX
        POP         EAX
        RET
delayRTC ENDP
SIZE_CSEG16     =       ($ - CODE16_BEGIN)
CODE16  ENDS

;-----------------------------------;
; 32-BIT PROTECTED MODE CODE SEGMENT ;
;-----------------------------------;

CODE32  SEGMENT PARA USE32
```

```
CODE32_BEGIN    =       $
        ASSUME          CS:CODE32,DS:DATA,ES:DATA

;--------------------;
; MAIN TASK (RING 0) ;
;--------------------;

ENTER_32:
        CALL    CLRSCR
        XOR     EDI, EDI

        LEA     ESI, MSG_TASK_MAIN
        CALL    W_ASCIIZ
        ADD     EDI, 80*2
        LEA     ESI, MSG_SCAN_CODE
        CALL    W_ASCIIZ
        ADD     EDI, 15*2

ENTER_32_WAIT_KEY:
        MOV     BL, KEY_SCAN_CODE
        MOVZX   EAX, BL
        CALL    W_EAX

        ADD     EDI, (80-15)*2
        MOV     EAX, ESP
        CALL    W_EAX                   ; VISUAL TEST OF ESP
        SUB     EDI, (80-15)*2

        CMP     BL, 1                   ; ESC?
        JNE     ENTER_32_WAIT_KEY       ; NOT ESC

;       MOV     ECX, 0FFFFFFFH
;       LOOP    $                       ; DOES A FEW SECOND DELAY ON MY
P233

        @RETF                           ; BACK TO 16-BIT CODE
```

```
;-------------------------------------------;
; MAIN ROUTINES AND HANDLERS ARE INCLUDED HERE ;
;-------------------------------------------;

        INCLUDE SCREENIO.INC                ; SCREEN I/O ROUTINES
        INCLUDE EXC.INC                     ; EXCEPTION HANDLER
        INCLUDE IRET.INC                    ; TIMER INTERRUPT HANDLER
        INCLUDE TIMER.INC                   ; KEYBOARD INTERRUPT
HANDLER
        INCLUDE KEYBOARD.INC                ; ROUTINE FOR DISPLAYING
TSS
        INCLUDE SHOW_TSS.INC                ; SERVICE INTERRUPT
        INCLUDE SERVICE.INC
SIZE_CSEG32     =       ($ - CODE32_BEGIN)
CODE32  ENDS

;---------------------------------------;
; COMMON REAL/PROTECTED MODE DATA SEGMENT ;
;---------------------------------------;

DATA    SEGMENT PARA USE16
DATA_BEGIN      =       $
        INCLUDE GDT.INC                     ; GDT TABLE
        INCLUDE IDT.INC                     ; IDT TABLE
        INCLUDE TSS.INC                     ; TSS TABLES

MSG_STARTUP     DB      "Tiny OS Kernel by Alexei A. Frounze (C)
2000",13,10,"$"
MSG_CONTACT     DB      "Contact information:",13,10
                DB      "E-mail  : alexfru@chat.ru",13,10
                DB      "Homepage: http://alexfru.chat.ru",13,10
                DB      "Mirror  : http://members.xoom.com/alexfru",13,10,"$"
MSG_VM  DB              "PROCESSOR IS ALREADY IN PROTECTED
MODE.",13,10
        DB              "PROBABLY WINDOWS OR EMM386.EXE IS
LOADED.",13,10
        DB              "LOAD PURE DOS OR WINDOWS IN COMMAND
```

```
PROMPT ONLY MODE.",13,10
        DB              "EXITING...",13,10,"$"
HEX_TAB DB              "0123456789ABCDEF"

EXC_ERR DB              0,0,0,0,0,0,0,0, 1,0,1,1,1,1,1,0, 0,1
MSG_EXC DB              "EXCEPTION: ",0
MSG_EXC_ERR DB          "ERROR CODE: ",0
MSG_CSEIP DB            "CS=XXXXXXXX EIP=XXXXXXXX",0

INT_MASK_M DB           ?
INT_MASK_S DB           ?
ESP32   DD              ?

TIMER_CNT DW            0
KEY_SCAN_CODE DB        0

MSG_TASK_MAIN DB        "MAIN TASK",0
MSG_SCAN_CODE DB        "KBD SCAN CODE: ",0
MSG_TASK_0 DB           "TASK #0",0
MSG_TASK_1 DB           "TASK #1",0

MSG_TSS LABEL BYTE
        DB "CR3  = XXXXXXXX FLAGS= XXXXXXXX",0
        DB "EIP  = XXXXXXXX CS   = XXXXXXXX",0
        DB "ESP  = XXXXXXXX SS   = XXXXXXXX",0
        DB "DS   = XXXXXXXX ES   = XXXXXXXX",0
        DB "FS   = XXXXXXXX GS   = XXXXXXXX",0
        DB "EAX  = XXXXXXXX EBX  = XXXXXXXX",0
        DB "ECX  = XXXXXXXX EDX  = XXXXXXXX",0
        DB "ESI  = XXXXXXXX EDI  = XXXXXXXX",0
        DB "EBP  = XXXXXXXX LINK = XXXXXXXX",0
        DB "ESP0 = XXXXXXXX SS0  = XXXXXXXX",0
        DB "ESP1 = XXXXXXXX SS1  = XXXXXXXX",0
        DB "ESP2 = XXXXXXXX SS2  = XXXXXXXX",0
        DB "LDTR = XXXXXXXX IO/T = XXXXXXXX",0

OFFS_TSS LABEL DWORD
```

```
        DD 1CH, 24H
        DD 20H, 4CH
        DD 38H, 50H
        DD 54H, 48H
        DD 58H, 5CH
        DD 28H, 34H
        DD 2CH, 30H
        DD 40H, 44H
        DD 3CH, 00H
        DD 04H, 08H
        DD 0CH, 10H
        DD 14H, 18H
        DD 60H, 64H
SIZE_DSEG       =       ($ - DATA_BEGIN)
DATA    ENDS

;---------------------------------;
; REAL/PROTECTED MODE STACK SEGMENT ;
;---------------------------------;

SSTACK  SEGMENT PARA STACK
        DB      SIZE_STACK DUP (?)
SSTACK  ENDS

;----------------------------;
; 2 ADDITIONAL TASKS (RING 3) ;
;----------------------------;

        INCLUDE TASKS.INC

END     START
```

gdt.inc (Описание GDT)

```
;-----------;
; GDT Table ;
;-----------;
```

```
GDT_BEGIN      =      $
GDT    LABEL          WORD
       GDT_0          S_DESC <0,0,0,0,0,0>                                    ; 00
       GDT_GDT        S_DESC <GDT_SIZE-
1,,,ACS_DATA,0,>                          ; 08
       GDT_CS16       S_DESC <SIZE_CSEG16-
1,,,ACS_CODE,0,>                          ; 10
       GDT_DS         S_DESC <SIZE_DSEG-
1,,,ACS_DATA+ACS_DPL_3,0,>                ; 18
       GDT_SS         S_DESC <SIZE_STACK-
1,,,ACS_DATA,0,>                          ; 20
       GDT_BIOS       S_DESC <SIZE_BIOS-
1,LOW_BIOS,HIGH_BIOS,ACS_DATA+ACS_DPL_3,0,0>  ; 28
       GDT_TEXT       S_DESC <SIZE_TEXT-
1,LOW_TEXT,HIGH_TEXT,ACS_DATA+ACS_DPL_3,0,0>  ; 30
       GDT_GRAPH      S_DESC <SIZE_GRAPH-
1,LOW_GRAPH,HIGH_GRAPH,ACS_DATA+ACS_DPL_3,0,0>;38
       GDT_FLAT       S_DESC <0FFFFH,0,0,ACS_DATA,08FH,0>    ; 4 GB
SEGMENT ;))      ; 40
       GDT_CS32       S_DESC <SIZE_CSEG32-
1,,,ACS_CODE+ACS_READ,0,>                 ; 48
       GDT_IDT        S_DESC <SIZE_IDT-1,,,ACS_IDT,0,>
; 50

       GDT_CS_0       S_DESC <SIZE_CS_0-
1,,,ACS_CODE+ACS_DPL_3,0,>                ; 58
       GDT_CS_1       S_DESC <SIZE_CS_1-
1,,,ACS_CODE+ACS_DPL_3,0,>                ; 60
       GDT_SS_0       S_DESC <SIZE_STACK-
1,,,ACS_DATA+ACS_DPL_3,0,>                ; 68
       GDT_SS_1       S_DESC <SIZE_STACK-
1,,,ACS_DATA+ACS_DPL_3,0,>                ; 70

       GDT_TSS_MAIN   S_DESC <SIZE_TSS-
1,,,ACS_TSS,0,>                           ; 78
       GDT_TSS_0      S_DESC <SIZE_TSS-
```

```
1,,,ACS_TSS,0,>                         ; 80
        GDT_TSS_1       S_DESC <SIZE_TSS-1,,,ACS_TSS,0,>
         ; 88
GDT_SIZE        =       ($ - GDT_BEGIN)

CS16_DESC       =           (GDT_CS16 - GDT_0)
DS_DESC         =           (GDT_DS - GDT_0)        + RPL_3
SS_DESC         =           (GDT_SS - GDT_0)
BIOS_DESC       =           (GDT_BIOS - GDT_0)      + RPL_3
TEXT_DESC       =           (GDT_TEXT - GDT_0)      + RPL_3
GRAPH_DESC      =           (GDT_GRAPH - GDT_0)     + RPL_3
FLAT_DESC       =           (GDT_FLAT - GDT_0)
CS32_DESC       =           (GDT_CS32 - GDT_0)
IDT_DESC        =           (GDT_IDT - GDT_0)
CS_0_DESC       =           (GDT_CS_0 - GDT_0)      + RPL_3
CS_1_DESC       =           (GDT_CS_1 - GDT_0)      + RPL_3
SS_0_DESC       =           (GDT_SS_0 - GDT_0)      + RPL_3
SS_1_DESC       =           (GDT_SS_1 - GDT_0)      + RPL_3
TSS_MAIN_DESC   =           (GDT_TSS_MAIN - GDT_0)
TSS_0_DESC      =           (GDT_TSS_0 - GDT_0)
TSS_1_DESC      =           (GDT_TSS_1 - GDT_0)
```

idt.inc (Описание IDT)

```
;-----------;
; IDT Table ;
;-----------;

IDTR    R_IDTR  <SIZE_IDT,0,0>
DUMMY_IDT       DF 0

IDT     LABEL           WORD
IDT_BEGIN       =       $
IRPC            N, 0123456789ABCDEF
        IDT_0&N         I_DESC <0, CS32_DESC, 0, ACS_TRAP, 0>   ; 00...0F
ENDM
IRPC            N, 0123456789ABCDEF
```

```
	IDT_1&N		I_DESC <0, CS32_DESC, 0, ACS_TRAP, 0>	; 10...1F
ENDM
	IDT_TIMER	I_DESC <0, CS32_DESC, 0, ACS_INT, 0>	; 20
	IDT_KEYBOARD	I_DESC <0, CS32_DESC, 0, ACS_INT, 0>	; 21
IRPC		N, 23456789ABCDEF
	IDT_2&N		I_DESC <0, CS32_DESC, 0, ACS_INT, 0>	; 22...2F
ENDM
	IDT_SERVICE	I_DESC <0, CS32_DESC, 0,
ACS_TRAP+ACS_DPL_3, 0>	; 30
SIZE_IDT	=	($ - IDT_BEGIN)
```

macroses.inc (Определение основных структур)

```
;-------------------------------------------------------------------;
; Macroses that define useful structures, constants and other things ;
;-------------------------------------------------------------------;

; Pushes several registers to the stack
@PUSH	MACRO	REGLIST
	IRP	REG, <REGLIST>
		PUSH	REG
	ENDM
ENDM

; Popes several registers from the stack
@POP	MACRO	REGLIST
	IRP	REG, <REGLIST>
		POP	REG
	ENDM
ENDM

; Sets up GDT table entry
@SET_GDT_ENTRY	MACRO
	MOV		[BX][S_DESC.BASE_L], AX
	MOV		[BX][S_DESC.BASE_M], DL
	MOV		[BX][S_DESC.BASE_H], DH
ENDM
```

```
; Segment Descriptor structure
S_DESC  STRUC
        LIMIT           DW      0
        BASE_L          DW      0
        BASE_M          DB      0
        ACCESS          DB      0
        ATTRIBS         DB      0
        BASE_H          DB      0
S_DESC  ENDS

; Interrupt Descriptor structure
I_DESC  STRUC
        OFFS_L          DW      0
        SEL             DW      0
        PARAM_CNT       DB      0
        ACCESS          DB      0
        OFFS_H          DW      0
I_DESC  ENDS

; IDTR register structure
R_IDTR  STRUC
        LIMIT           DW      0
        IDT_L           DW      0
        IDT_H           DW      0
R_IDTR  ENDS

; Task State Segment structure
S_TSS   STRUC
        LINK            DW      0, 0    ;  0
        ESP0            DD      0       ;  4
        SS0             DW      0, 0    ;  8
        ESP1            DD      0       ; 0C
        SS1             DW      0, 0    ; 10
        ESP2            DD      0       ; 14
        SS2             DW      0, 0    ; 18
        R_CR3           DD      0       ; 1C
```

```
        R_EIP           DD      0       ; 20
        R_EFLAGS        DD      0       ; 24
        R_EAX           DD      0       ; 28
        R_ECX           DD      0       ; 2C
        R_EDX           DD      0       ; 30
        R_EBX           DD      0       ; 34
        R_ESP           DD      0       ; 38
        R_EBP           DD      0       ; 3C
        R_ESI           DD      0       ; 40
        R_EDI           DD      0       ; 44
        R_ES            DW      0, 0    ; 48
        R_CS            DW      0, 0    ; 4C
        R_SS            DW      0, 0    ; 50
        R_DS            DW      0, 0    ; 54
        R_FS            DW      0, 0    ; 58
        R_GS            DW      0, 0    ; 5C
        R_LDTR          DW      0, 0    ; 60
        TRACE           DW      0       ; 64
        IO_MAP_ADDR     DW      68H     ; 66
        IO_MAP          DB      (400H SHR 3) DUP (0) ; 400H PORTS
AVAILABLE
S_TSS   ENDS

; DIFFERENT CONSTANTS AND FLAGS FOR PM
SIZE_TSS        EQU     68H + (400H SHR 3)

; Access byte's flags
ACS_PRESENT     EQU             10000000B
ACS_CSEG        EQU             00011000B
ACS_DSEG        EQU             00010000B
ACS_EXPDOWN     EQU             00000100B
ACS_CONFORM     EQU             00000100B
ACS_READ        EQU             00000010B
ACS_WRITE       EQU             00000010B

ACS_CODE        =               ACS_PRESENT OR ACS_CSEG; OR
ACS_CONFORM
```

```
ACS_DATA        =               ACS_PRESENT OR ACS_DSEG OR
ACS_WRITE
ACS_STACK       =               ACS_PRESENT OR ACS_DSEG OR
ACS_WRITE OR ACS_EXPDOWN

ACS_INT_GATE    EQU             00001110B
ACS_TRAP_GATE   EQU             00001111B
ACS_IDT         EQU             ACS_DATA
ACS_INT         EQU             ACS_PRESENT OR ACS_INT_GATE
ACS_TRAP        EQU             ACS_PRESENT OR ACS_TRAP_GATE
ACS_TSS         EQU             ACS_PRESENT OR 00001001B

ACS_DPL_0       EQU             00000000B
ACS_DPL_1       EQU             00100000B
ACS_DPL_2       EQU             01000000B
ACS_DPL_3       EQU             01100000B

RPL_0           EQU             0
RPL_1           EQU             1
RPL_2           EQU             2
RPL_3           EQU             3

; CONSTANTS FOR BIOS DATA AREA
SEG_BIOS        EQU             00040H
SIZE_BIOS       EQU             00300H
LOW_BIOS        EQU             00400H
HIGH_BIOS       EQU             0

; CONSTANTS FOR TEXT VIDEO MODES
SEG_TEXT        EQU             0B800H
SIZE_TEXT       EQU             02000H          ; > 80*50*2
LOW_TEXT        EQU             08000H
HIGH_TEXT       EQU             0BH

; CONSTANTS FOR GRAPHICS VIDEO MODES
SEG_GRAPH       EQU             0A000H
SIZE_GRAPH      EQU             10000H
```

```
LOW_GRAPH       EQU             0
HIGH_GRAPH      EQU             0AH

; DIFFERENT CONSTANTS FOR PORT I/O
PORT_CMOS       EQU             070H
PORT_6845       EQU             00063H
PORT_TEXT       EQU             003D4H
PORT_STATUS     EQU             064H
SHUT_DOWN       EQU             0FEH
MODE_VIRTUAL    EQU             00001H
PORT_A20        EQU             0D1H
A20_ON          EQU             0DFH
A20_OFF         EQU             0DDH
PORT_KBD_A      EQU             060H
PORT_KBD_B      EQU             061H
PORT_INT_MASK_M EQU             021H
PORT_INT_MASK_S EQU             0A1H

EOI             EQU             020H
PORT_8259M      EQU             020H
PORT_8259S      EQU             0A0H

PORT_COM_REG    EQU             043H
PORT_CHANNEL2   EQU             042H

; Enables A20 line
@ENABLE_A20     MACRO
        MOV             AL, PORT_A20
        OUT             PORT_STATUS, AL
        MOV             AL, A20_ON
        OUT             PORT_KBD_A, AL
ENDM

; Disables A20 line
@DISABLE_A20    MACRO
        MOV             AL, PORT_A20
        OUT             PORT_STATUS, AL
```

```
        MOV             AL, A20_OFF
        OUT             PORT_KBD_A, AL
ENDM

; Enables non-maskable interrupts
@ENABLE_NMI     MACRO
        MOV             AL, 0DH
        OUT             PORT_CMOS, AL
        JMP             $+2
ENDM

; Disables non-maskable interrupts
@DISABLE_NMI    MACRO
        MOV             AL, 8FH
        OUT             PORT_CMOS, AL
        JMP             $+2
ENDM

; This macro reprograms PIC (master and slave) to other interrupt vectors
@SET_INT_CTRLR  MACRO   INT_MASTER, INT_SLAVE
        MOV     AL, 11H                 ; START 8259 INITIALIZATION
        OUT     PORT_8259M, AL
        OUT     PORT_8259S, AL
        MOV     AL, INT_MASTER          ; BASE INTERRUPT VECTOR
        OUT     PORT_8259M+1, AL
        MOV     AL, INT_SLAVE
        OUT     PORT_8259S+1, AL
        MOV     AL, 1 SHL 2             ; BITMASK FOR CASCADE ON IRQ 2
        OUT     PORT_8259M+1, AL
        MOV     AL, 2                   ; CASCADE ON IRQ 2
        OUT     PORT_8259S+1, AL
        MOV     AL, 1                   ; FINISH 8259 INITIALIZATION
        OUT     PORT_8259M+1, AL
        OUT     PORT_8259S+1, AL

        MOV     AL, 0FFH                ; MASK ALL INTERRUPTS
        OUT     PORT_INT_MASK_M, AL
```

```
        OUT     PORT_INT_MASK_S, AL
ENDM

@JUMP   MACRO           ; OVERLOADS CODE SELECTOR AFTER
CHANGING ITS DESCRIPTOR
        DB      0EAH
        DW      $+4
        DW      CS16_DESC
ENDM

@JUMPR  MACRO           ; OVERLOADS CODE SEGMENT
        DB      0EAH
        DW      $+4
        DW      CODE16
ENDM

@RETF   MACRO           ; FAR RETURN FROM 16-BIT TO 32-BIT CODE &
VICE VERSA
        DB      66H
        RETF
ENDM
```

tss.inc (Описание TSS)

```
;-------------------------------------;
; TSSs of main and few additional tasks ;
;-------------------------------------;

TSS_MAIN        S_TSS   <>
TSS_0           S_TSS   <>
TSS_1           S_TSS   <>

; TASK LIST
TASK_LIST       DW      TSS_MAIN_DESC, TSS_0_DESC, TSS_1_DESC, 0

; THIS INDEX VARIABLE IS USED TO CHOSE THE TASK TO SWITCH
TO
```

```
TASK_INDEX      DW      2

; THIS 6-BYTE PM ADDRESS IS USED TO PERFORM FAR JUMP TO TSS
TASK_ADDR       DF      0
```

iret.inc (Программирование контроллера прерываний)

```
;-------------------------------------;
; Dummy IRets for hardware interrupts ;
;-------------------------------------;

DUMMY_IRET0 PROC NEAR
        PUSH    EAX
        MOV     AL, EOI
        OUT     PORT_8259M, AL
        POP     EAX
        IRETD
DUMMY_IRET0 ENDP

DUMMY_IRET1 PROC NEAR
        PUSH    EAX
        MOV     AL, EOI
        OUT     PORT_8259M, AL
        OUT     PORT_8259S, AL
        POP     EAX
        IRETD
DUMMY_IRET1 ENDP
```

keyboard.inc (Обработчик прерывания от клавиатуры)

```
;-------------------------------------;
; Keyboard hardware interrupt handler ;
;-------------------------------------;
```

```
KEYBOARD_HANDLER PROC NEAR
        @PUSH   <DS, EAX>
        MOV     AX, DS_DESC
        MOV     DS, AX

        IN      AL, PORT_KBD_A
        MOV     AH, AL
        AND     AL, 07FH

        MOV     DS:[KEY_SCAN_CODE], AL

;       IN      AL, PORT_KBD_B              ; ¦      0
;       OR      AL, 080H                  ; ¦
;       OUT     PORT_KBD_B, AL              ; L----¬
;       IN      AL, PORT_KBD_B              ;     ¦ 1
;       AND     AL, 07FH                    ;     ¦
;       OUT     PORT_KBD_B, AL              ; ------
;                                   ; ¦      0
        MOV     AL, EOI
        OUT     PORT_8259M, AL
        @POP    <EAX, DS>
        IRETD
KEYBOARD_HANDLER ENDP
```

timer.inc (Обработчик прерывания от таймера)

```
;--------------------------------;
; Timer hardware interrupt handler ;
;--------------------------------;

TIMER_HANDLER PROC NEAR
        @PUSH   <DS, ES>
        PUSHAD
        MOV     AX, DS_DESC
        MOV     DS, AX
        MOV     AX, BIOS_DESC
        MOV     ES, AX
```

```
    INC     DWORD PTR ES:[06CH]           ; UPDATE BIOS TICK
COUNTER

    MOV     AX, DS:[TIMER_CNT]            ; INTERNAL TICK COUNTER
    INC     AX
    CMP     AX, 9                   ; IS TIME FOR "*" OR " "?
    JNE     TIMER_END                   ; NO, IT ISN'T

    MOV     AX, TEXT_DESC
    MOV     ES, AX
    MOV     AL, ES:[2*79]
    MOV     BYTE PTR ES:[2*79], 0         ; " " <-> "*"
    CMP     AL, '*'
    JE      TIMER_FLAG
    MOV     BYTE PTR ES:[2*79], '*'       ; "*" <-> " "

TIMER_FLAG:
    XOR     AX, AX
TIMER_END:                          ; UPDATE INTERNAL TICK
COUNTER
    MOV     DS:[TIMER_CNT], AX

    MOV     AL, EOI
    OUT     PORT_8259M, AL              ; ACKNOWLEDGE HANDLING
OF INTERRUPT

; LOOKING FOR THE NEXT TASK BEING CONTINUED

    MOVZX   EAX, DS:[TASK_INDEX]
    ADD     DS:[TASK_INDEX], 2
    MOVZX   EAX, DS:[TASK_LIST+EAX]
    OR      EAX, EAX                    ; END OF TASK LIST?
    JNZ     TIMER_NEXT_TASK             ; NOT END, RUNNING NEXT
TASK
    MOVZX   EAX, DS:[TASK_LIST]
    MOV     DS:[TASK_INDEX], 2          ; END, RUNNING FIRST TASK
```

```
TIMER_NEXT_TASK:
	MOV	WORD PTR DS:[TASK_ADDR+4], AX	; FAR POINTER TO
TSS

	JMP	FWORD PTR DS:[TASK_ADDR]	; SWITCH TO ANOTHER
TASK :)

	POPAD
	@POP	<ES, DS>
	IRETD
TIMER_HANDLER ENDP
```

screenio.inc (Работа с видеокартой)

```
;--------------------------------;
; Subroutines for screen text I/O ;
;--------------------------------;

; FUNCTION: CLEARS THE SCREEN
; IN:  NONE
; OUT: NONE, ALL REGS ARE SAVED
CLRSCR  PROC NEAR
	PUSH	ES
	PUSHAD
	MOV	AX, TEXT_DESC
	MOV	ES, AX
	XOR	EDI, EDI
	MOV	ECX, 80*25
	MOV	AX, 700H;1B00H
	REP	STOSW
	POPAD
	POP	ES
	RET
CLRSCR  ENDP

; FUNCTION: PRINTS THE EAX VALUE TO THE SCREEN
; IN:  EAX = 32-BIT VALUE, EDI = SCREEN OFFSET
```

```
; OUT: NONE, ALL REGS ARE SAVED
W_EAX   PROC NEAR
        @PUSH    <DS,ES>
        PUSHAD
        MOV     DX, DS_DESC
        MOV     DS, DX
        MOV     DX, TEXT_DESC
        MOV     ES, DX
        MOV     ECX, 8
        LEA     ESI, HEX_TAB
W_EAX_1:
        ROL     EAX, 4
        MOVZX   EBX, AL
        AND     BL, 0FH
        MOV     BL, DS:[ESI+EBX]
        MOV     ES:[EDI], BL
        ADD     EDI, 2
        LOOP    W_EAX_1
        POPAD
        @POP    <ES,DS>
        RET
W_EAX   ENDP

; FUNCTION: PRINTS ASCIIZ STRING TO THE SCREEN
; IN:  DS:ESI -> STRING, EDI = SCREEN OFFSET
; OUT: NONE, ALL REGS ARE SAVED
W_ASCIIZ PROC NEAR
        PUSH    ES
        PUSHAD
        MOV     AX, TEXT_DESC
        MOV     ES, AX
W_ASCIIZ_1:
        LODSB
        OR      AL, AL
        JZ      W_ASCIIZ_2
        STOSB
        INC     EDI
```

```
        JMP     W_ASCIIZ_1
W_ASCIIZ_2:
        POPAD
        POP     ES
        RET
W_ASCIIZ ENDP
```

service.inc (Формирование обработчика прерывания)

```
;---------------------------------;
; SERVICE SOFTWARE INTERRUPT (30H) ;
;---------------------------------;

SERVICE_INTERRUPT PROC NEAR
        PUSHAD
        AND     EBX, 0FF00H                 ; BH = FUNCTION CODE
        SHR     EBX, 8
        CMP     EBX, 3
        JNC     SERV_INT_END                ; ONLY 3 FUNCTIONS
AVAILABLE
        SHL     EBX, 2                      ; ADDRESS OFFSET TABLE
        CALL    DWORD PTR CS:[FUNC_TAB+EBX]
SERV_INT_END:
        POPAD
        IRETD

F_00 LABEL NEAR
        JMP     CLRSCR                      ; CLEAR SCREEN

F_01 LABEL NEAR
        JMP     W_ASCIIZ                    ; OUT ASCIIZ STRING

F_02 LABEL NEAR
        JMP     W_EAX                       ; OUT 32-BIT EAX AS HEX
```

```
FUNC_TAB LABEL DWORD
        DD      F_00, F_01, F_02
SERVICE_INTERRUPT ENDP
```

set_rm1.inc (Возврат в RealMode – способ 1)

```
;-------------------------------------------------------------------------;
; Sets Real Mode using CR0, overloading segs and reprogramming PIC        ;
; Ver 1: Unfortunately, works quite buggy. I don't know why. :((          ;
; Ver 2: It seems to be everything is OK after changing order of the      ;
;         @SET_INT_CTRLR 08H, 70H and                                     ;
;         LIDT FWORD PTR IDTR                                             ;
;       instructions                                                      ;
; Ver 3: unchanged vesrsion 2, but it doesn't work again when the task    ;
;       switching is enabled                                              ;
; Ver 4: added a "CLTS" instructio. Every DPMI-based program, that I tried ;
;       to run immediately after quitting from this program, just hanged  ;
;       with nonzero "task switched" flag. The solution is inside this    ;
;       simple instruction. Viola! it works!                              ;
;-------------------------------------------------------------------------;

        CLTS                            ; THE SOLUTION!!! ;-)

        @SET_INT_CTRLR 08H, 70H                 ; REINITIALISE MASTER &
SLAVE INT CONTROLLERS

        MOV     GDT_CS16.LIMIT, 0FFFFH          ; CODE SEGMENT HAS
64KB LIMIT
        MOV     GDT_DS.LIMIT, 0FFFFH            ; DATA SEGMENT HAS 64KB
LIMIT
        MOV     GDT_SS.LIMIT, 0FFFFH            ; STACK SEGMENT HAS
64KB LIMIT
;       LGDT    FWORD PTR GDT_GDT               ; LOAD GDTR

        @JUMP                           ; OVERLOAD CODE SELECTOR
        MOV     AX, DS_DESC
        MOV     DS, AX                      ; OVERLOAD DATA SELECTOR
```

```
        MOV     ES, AX                  ; OVERLOAD DATA SELECTOR
        MOV     FS, AX                  ; OVERLOAD DATA SELECTOR
        MOV     GS, AX                  ; OVERLOAD DATA SELECTOR
        MOV     AX, SS_DESC
        MOV     SS, AX                  ; OVERLOAD STACK SELECTOR

        MOV     EAX, CR0
        AND     AL, 0FEH
        MOV     CR0, EAX                ; BACK TO THE REAL MODE

        @JUMPR                          ; RESTORE RM CODE SEGMENT
        MOV     AX, SSTACK
        MOV     SS, AX                  ; RESTORE RM STACK SEGMENT
        MOV     AX, DATA
        MOV     DS, AX                  ; RESTORE RM DATA SEGMENT
        MOV     ES, AX
        XOR     AX, AX
        MOV     FS, AX
        MOV     GS, AX

        MOV     IDTR.LIMIT, 3FFH        ; REAL MODE INTERRUPT
TABLE SIZE
        MOV     DWORD PTR IDTR+2, 0     ; REAL MODE INTERRUPT
TABLE ADDRESS
        LIDT    FWORD PTR IDTR          ; LOAD REAL MODE INT
TABLE
```

set_rm2.inc (Возврат в RealMode – способ 2)

```
;------------------------------;
; Sets Real Mode using CPU reset ;
; Works perfectly. :))          ;
;------------------------------;

        MOV     ESP32, ESP
        MOV     AX, BIOS_DESC
        MOV     ES, AX
```

```
        MOV     WORD PTR ES:[067H], OFFSET SHUTDOWN_RET
        MOV     WORD PTR ES:[069H], CODE16
        MOV     AL, 5
        OUT     PORT_CMOS+1, AL

;       MOV     AL, SHUT_DOWN
;       OUT     PORT_STATUS, AL
;START_RESET:
;       HLT
;       JMP     START_RESET
        LIDT    [DUMMY_IDT]
        INT     0

SHUTDOWN_RET    LABEL FAR
        MOV     AX, DATA
        MOV     DS, AX
        MOV     ES, AX
        MOV     AX, SSTACK
        MOV     SS, AX
        MOV     ESP, ESP32
        XOR     AX, AX
        MOV     FS, AX
        MOV     GS, AX
```

setupgdt.inc (Инициализация GDT)

```
;-----------;
; Fills GDT ;
;-----------;

        MOV     AX, DATA
        MOV     DL, AH
        XOR     DH, DH
        SHL     AX, 4
        SHR     DX, 4
        MOV     SI, AX
        MOV     DI, DX
```

```
; GDT GOES HERE

    LEA     BX, GDT_GDT
    MOV     AX, SI
    MOV     DX, DI
    ADD     AX, OFFSET GDT
    ADC     DX, 0
    @SET_GDT_ENTRY

; 16-BIT REAL/PROTECTED MODE MAIN TASK CODE GOES HERE

    LEA     BX, GDT_CS16
    MOV     AX, CS
    XOR     DH, DH
    MOV     DL, AH
    SHL     AX, 4
    SHR     DX, 4
    @SET_GDT_ENTRY

; COMMON DATA SEGMENT GOES HERE

    LEA     BX, GDT_DS
    MOV     AX, SI
    MOV     DX, DI
    @SET_GDT_ENTRY

; STACK FOR 16/32 REAL/PROTECTED MODE MAIN TASK GOES HERE

    LEA     BX, GDT_SS
    MOV     AX, SS
    XOR     DH, DH
    MOV     DL, AH
    SHL     AX, 4
    SHR     DX, 4
    @SET_GDT_ENTRY
```

```
; 32-BIT PROTECTED MODE MAIN TASK CODE GOES HERE

        LEA     BX, GDT_CS32
        MOV     AX, CODE32
        XOR     DH, DH
        MOV     DL, AH
        SHL     AX, 4
        SHR     DX, 4
        @SET_GDT_ENTRY
        OR      [BX][S_DESC.ATTRIBS], 40H          ; 32-BIT STUFF

; IDT GOES HERE

        LEA     BX, GDT_IDT
        MOV     AX, SI
        MOV     DX, DI
        ADD     AX, OFFSET IDT
        ADC     DX, 0
        @SET_GDT_ENTRY
        MOV     IDTR.IDT_L, AX
        MOV     IDTR.IDT_H, DX

; TASKS' CODES GO HERE

        LEA     BX, GDT_CS_0
        MOV     AX, CODE_0
        XOR     DH, DH
        MOV     DL, AH
        SHL     AX, 4
        SHR     DX, 4
        @SET_GDT_ENTRY
        OR      [BX][S_DESC.ATTRIBS], 40H          ; 32-BIT STUFF

        LEA     BX, GDT_CS_1
        MOV     AX, CODE_1
        XOR     DH, DH
        MOV     DL, AH
```

```
        SHL     AX, 4
        SHR     DX, 4
        @SET_GDT_ENTRY
        OR      [BX][S_DESC.ATTRIBS], 40H           ; 32-BIT STUFF

; TASKS' STACKS GO HERE

        LEA     BX, GDT_SS_0
        MOV     AX, STCK_0
        XOR     DH, DH
        MOV     DL, AH
        SHL     AX, 4
        SHR     DX, 4
        @SET_GDT_ENTRY

        LEA     BX, GDT_SS_1
        MOV     AX, STCK_1
        XOR     DH, DH
        MOV     DL, AH
        SHL     AX, 4
        SHR     DX, 4
        @SET_GDT_ENTRY

; TSSs GO HERE

        LEA     BX, GDT_TSS_MAIN
        MOV     AX, SI
        MOV     DX, DI
        ADD     AX, OFFSET TSS_MAIN
        ADC     DX, 0
        @SET_GDT_ENTRY

        LEA     BX, GDT_TSS_0
        MOV     AX, SI
        MOV     DX, DI
        ADD     AX, OFFSET TSS_0
        ADC     DX, 0
```

```
        @SET_GDT_ENTRY

        LEA     BX, GDT_TSS_1
        MOV      AX, SI
        MOV      DX, DI
        ADD      AX, OFFSET TSS_1
        ADC      DX, 0
        @SET_GDT_ENTRY
```

setupidt.inc (Инициализация IDT)

```
;-----------;
; Fills IDT ;
;-----------;

IRPC            N, 0123456789ABCDEF                 ; 00...0F
        LEA     EAX, EXC_0&N
        MOV      IDT_0&N.OFFS_L, AX
        SHR     EAX, 16
        MOV      IDT_0&N.OFFS_H, AX
ENDM
IRPC            N, 0123456789ABCDEF                 ; 10...1F
        LEA     EAX, EXC_1&N
        MOV      IDT_1&N.OFFS_L, AX
        SHR     EAX, 16
        MOV      IDT_1&N.OFFS_H, AX
ENDM

        LEA     EAX, TIMER_HANDLER                    ; 20
        MOV      IDT_TIMER.OFFS_L, AX
        SHR     EAX, 16
        MOV      IDT_TIMER.OFFS_H, AX

        LEA     EAX, KEYBOARD_HANDLER                   ; 21
        MOV      IDT_KEYBOARD.OFFS_L, AX
        SHR     EAX, 16
        MOV      IDT_KEYBOARD.OFFS_H, AX
```

```
IRPC        N, 234567                   ; 22...27
        LEA     EAX, DUMMY_IRET0
        MOV     IDT_2&N.OFFS_L, AX
        SHR     EAX, 16
        MOV     IDT_2&N.OFFS_H, AX
ENDM
IRPC        N, 89ABCDEF                 ; 28...2F
        LEA     EAX, DUMMY_IRET1
        MOV     IDT_2&N.OFFS_L, AX
        SHR     EAX, 16
        MOV     IDT_2&N.OFFS_H, AX
ENDM

        LEA     EAX, SERVICE_INTERRUPT          ; 30
        MOV     IDT_SERVICE.OFFS_L, AX
        SHR     EAX, 16
        MOV     IDT_SERVICE.OFFS_H, AX
```

setuptss.inc (Формирование TSS)

```
;-------------------------------;
; Fills TSSs of additional tasks ;
;-------------------------------;

        OR      TSS_0.R_EFLAGS, 3200H       ; IOPL=3, ENABLED
INTERRUPTS
        MOV     TSS_0.R_CS, CS_0_DESC
        MOV     TSS_0.R_EIP, OFFSET TASK_0
        MOV     TSS_0.R_SS, SS_0_DESC
        MOV     TSS_0.R_ESP, SIZE_STACK
        MOV     AX, 0
        MOV     TSS_0.R_DS, AX
        MOV     TSS_0.R_ES, AX
        MOV     TSS_0.R_FS, AX
        MOV     TSS_0.R_GS, AX
        MOV     TSS_0.SS0, SS_DESC
```

```
        MOV     TSS_0.ESP0, SIZE_STACK SHR 1

        OR      TSS_1.R_EFLAGS, 3200H           ; IOPL=3, ENABLED
INTERRUPTS
        MOV     TSS_1.R_CS, CS_1_DESC
        MOV     TSS_1.R_EIP, OFFSET TASK_1
        MOV     TSS_1.R_SS, SS_1_DESC
        MOV     TSS_1.R_ESP, SIZE_STACK
        MOV     AX, 0
        MOV     TSS_1.R_DS, AX
        MOV     TSS_1.R_ES, AX
        MOV     TSS_1.R_FS, AX
        MOV     TSS_1.R_GS, AX
        MOV     TSS_1.SS0, SS_DESC
        MOV     TSS_1.ESP0, SIZE_STACK SHR 2
```

show_tss.inc (Просмотр)

```
; IN: EAX = TSS OFFSET
SHOW_TSS PROC NEAR
        PUSH    DS
        PUSHAD
        MOV     EBX, EAX
        MOV     AX, DS_DESC
        MOV     DS, AX

        MOV     EDI, 10*80*2
        LEA     ESI, MSG_TSS
        LEA     EDX, OFFS_TSS
        MOV     ECX, 13
SHOW_TSS_CYCLE:
        CALL    W_ASCIIZ
        ADD     ESI, 32

        ADD     EBX, DS:[EDX]
        MOV     EAX, DS:[EBX]
        SUB     EBX, DS:[EDX]
```

```
        ADD     EDI, 7*2
        CALL    W_EAX
        ADD     EDX, 4

        ADD     EBX, DS:[EDX]
        MOV     EAX, DS:[EBX]
        SUB     EBX, DS:[EDX]
        ADD     EDI, 16*2
        CALL    W_EAX
        ADD     EDX, 4

        ADD     EDI, (80-7-16)*2
        LOOP    SHOW_TSS_CYCLE
        POPAD
        POP     DS
        RET
SHOW_TSS ENDP
```

tasks.inc (Описание задач)

```
;----------------------------;
; 2 ADDITIONAL TASKS (RING 3) ;
;----------------------------;

;---------------------;
; TASK #0 CODE SEGMENT ;
;---------------------;

CODE_0  SEGMENT PARA USE32
        ASSUME  CS:CODE_0
CS_0_BEGIN      =       $
TASK_0  PROC NEAR
        MOV     AX, DS_DESC
        MOV     DS, AX
        MOV     ES, AX
        LEA     ESI, MSG_TASK_0
        MOV     EDI, 5*80*2
```

```
        MOV     BH, 1
        INT     30H
        MOV     EAX, 0
        ADD     EDI, 80*2
TASK_0_L:
        MOV     BH, 2
        INT     30H
        INC     EAX

        MOV     EDX, EAX

        ADD     EDI, 80*2
        MOV     EAX, ESP
        MOV     BH, 2
        INT     30H                     ; VISUAL TEST OF ESP

        SUB     EDI, 80*2
        MOV     EAX, EDX

        JMP     TASK_0_L
TASK_0  ENDP
SIZE_CS_0       =       ($ - CS_0_BEGIN)
CODE_0  ENDS

;---------------------;
; TASK #1 CODE SEGMENT ;
;---------------------;

CODE_1  SEGMENT PARA USE32
        ASSUME  CS:CODE_1
CS_1_BEGIN      =       $
TASK_1  PROC NEAR
        MOV     AX, DS_DESC
        MOV     DS, AX
        MOV     ES, AX
        LEA     ESI, MSG_TASK_1
        MOV     EDI, 5*80*2+10*2
```

```
        MOV     BH, 1
        INT     30H
        MOV     EAX, -1
        ADD     EDI, 80*2
TASK_1_L:
        MOV     BH, 2
        INT     30H
        DEC     EAX

        MOV     EDX, EAX

        ADD     EDI, 80*2
        MOV     EAX, ESP
        MOV     BH, 2
        INT     30H                     ; VISUAL TEST OF ESP

        SUB     EDI, 80*2
        MOV     EAX, EDX

        JMP     TASK_1_L
TASK_1  ENDP
SIZE_CS_1       =       ($ - CS_1_BEGIN)
CODE_1  ENDS

;----------------------;
; TASK #0 STACK SEGMENT ;
;----------------------;

STCK_0  SEGMENT PARA
        DB      SIZE_STACK DUP (?)
STCK_0  ENDS

;----------------------;
; TASK #1 STACK SEGMENT ;
;----------------------;

STCK_1  SEGMENT PARA
```

```
        DB      SIZE_STACK DUP (?)
STCK_1  ENDS
```

exc.inc

```
;---------------------;
; Exception handler(s) ;
;---------------------;

M       =       0
IRPC            N, 0123456789ABCDEF
EXC_0&N LABEL WORD
        CLI
        PUSH            EBP
        MOV             EBP, ESP
        PUSHAD
        MOV             EAX, M
        M       =       M+1
        JMP             EXC_HANDLER
ENDM

M       =       010H
IRPC            N, 0123456789ABCDEF
EXC_1&N LABEL WORD
        CLI
        PUSH            EBP
        MOV             EBP, ESP
        PUSHAD
        MOV             EAX, M
        M       =       M+1
        JMP             EXC_HANDLER
ENDM

EXC_HANDLER     PROC    NEAR
        PUSH    DS

        MOV     BX, DS_DESC
```

```
        MOV     DS, BX

;       CALL    CLRSCR
        LEA     ESI, MSG_EXC
        MOV     EDI, 40*2
        CALL    W_ASCIIZ                ; "EXCEPTION: "
        ADD     EDI, 11*2
        CALL    W_EAX                   ; EXCEPTION NUMBER
        SUB     EDI, 11*2

        MOV     EBX, EBP
        ADD     EBX, 4                  ; EBX = ^CS:EIP

        MOVZX   EAX, BYTE PTR DS:[EXC_ERR+EAX]
        OR      EAX, EAX
        JZ      EXC_HANDLER_NO_ERR_CODE         ; THE EXCEPTION HAS
NO ERROR CODE

        ADD     EBX, 4                  ; CORRECT EBX

        LEA     ESI, MSG_EXC_ERR
        ADD     EDI, 80*2
        CALL    W_ASCIIZ                ; "ERROR CODE: "

        MOV     EAX, SS:[EBP+4]
        AND     EAX, 0FFFFH
        ADD     EDI, 12*2
        CALL    W_EAX
        SUB     EDI, 12*2
        SUB     EDI, 80*2

EXC_HANDLER_NO_ERR_CODE:
        LEA     ESI, MSG_CSEIP
        ADD     EDI, 2*80*2
        CALL    W_ASCIIZ                ; "CS=XXXXXXXX
EIP=XXXXXXXX"
        ADD     EDI, 3*2
```

```
        MOVZX   EAX, WORD PTR SS:[EBX+4]
        CALL    W_EAX                   ; CS
        ADD     EDI, 13*2
        MOV     EAX, SS:[EBX]
        CALL    W_EAX                   ; EIP
        SUB     EDI, (3+13)*2

;       jmp     $                       ; INFINITE LOOP

        MOV     AX, SS_DESC
        MOV     SS, AX
        MOV     ESP, DS:[ESP32]
        @RETF

        IRETD
EXC_HANDLER     ENDP
```